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

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

  - Serializability is the classical concurrency scheme.

  - It ensures that a schedule for executing concurrent transactions is equivalent to one that executes the transactions serially in some order.

- **Serializable schedule**

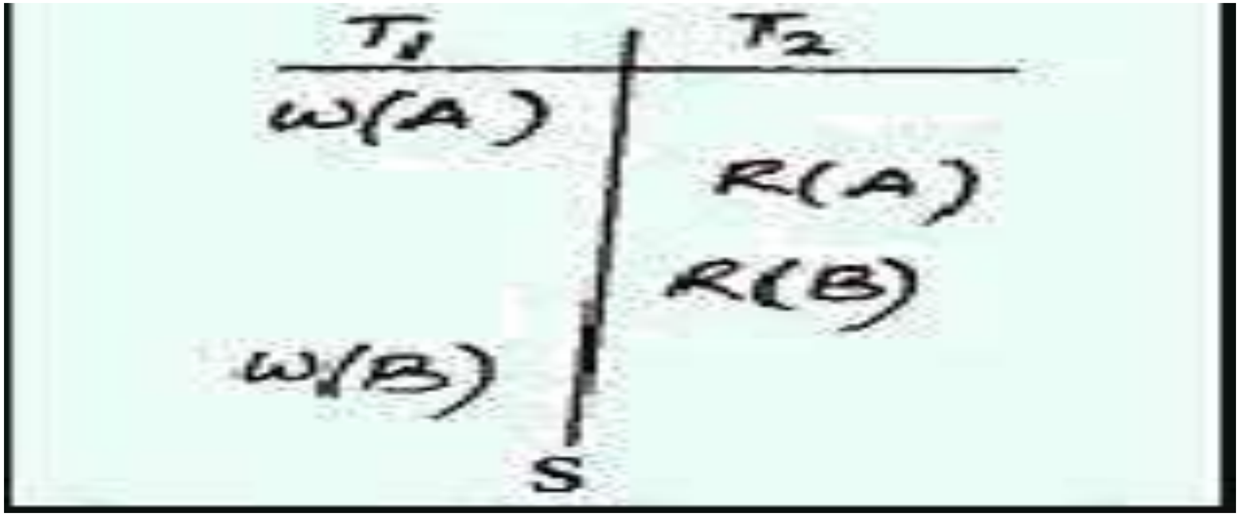
- If a schedule is equivalent to some serial schedule then that schedule is called Serializable schedule

- Let us consider a schedule S. What the schedule S says ?

- Read A after updation.

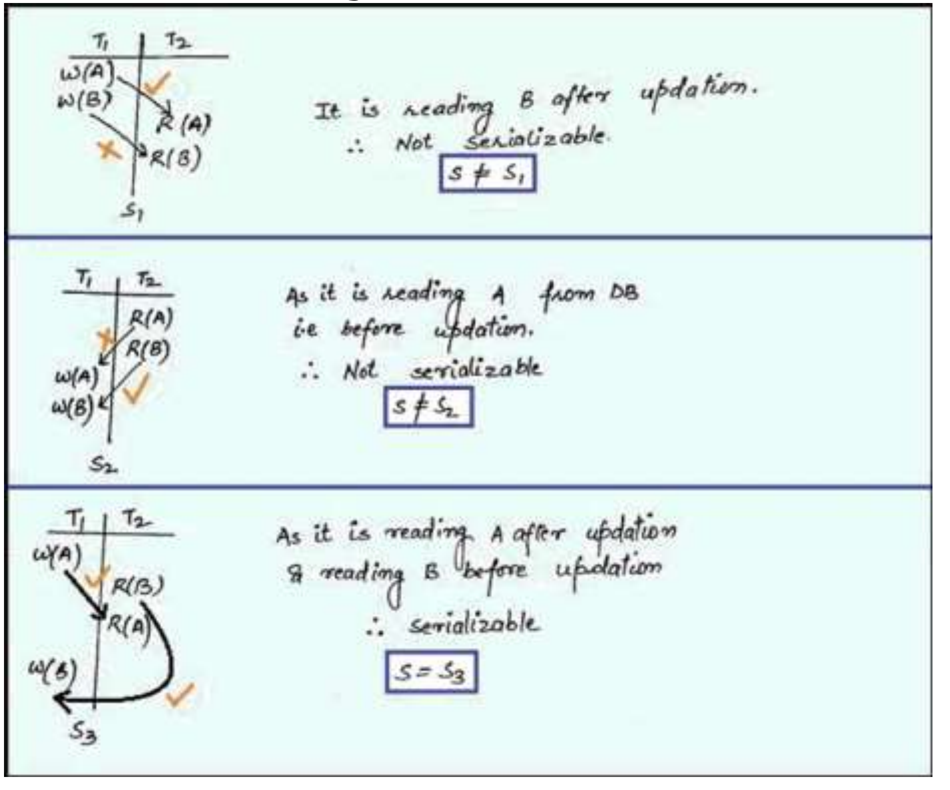
- Read B before updation.

SERIALIZABILITY



SERIALIZABILITY

- Let us consider 3 schedules S1, S2, and S3. We have to check whether they are serializable with S or not ?



SERIALIZABILITY

- Types of Serializability
  - -Conflict Serializability
  - -View Serializability

- **Conflict Serializable**
- Any given concurrent schedule is said to be Conflict Serializable if and only if it is Conflict equivalent to one of the possible serial schedule.
- Two schedules would be conflicting if they have the following properties
  - Both belong to separate transactions.
  - Both accesses the same data item.
  - At least one of them is "write" operation.

SERIALIZABILITY

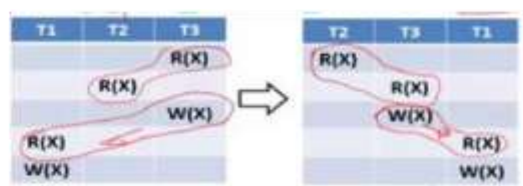
- **Conflicting Instructions**
- Instructions l_i and l_j of transactions T_i and T_j respectively, **conflict** if they are operations by different transaction on the same data item, and at least one of these instruction is **write** operation.
  1. l_i = **read**(Q), l_j = **read**(Q). l_i and l_j don't conflict.
  2. l_i = **read**(Q), l_j = **write**(Q). They conflict.
  3. l_i = **write**(Q), l_j = **read**(Q). They conflict
  4. l_i = **write**(Q), l_j = **write**(Q). They conflict
- Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if
    - Both the schedules contain the same set of Transactions.
    - The order of conflicting pairs of operation is maintained in both the schedules.
  - 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**.
  - We say that a schedule S is **conflict serializable** if it is conflict equivalent to a serial schedule
- Schedule 3 can be transformed into Schedule 6, a serial schedule where T_2 follows T_1, by series of swaps of non-conflicting instructions. Therefore Schedule 3 is conflict serializable.
- **Schedule 3**

SERIALIZABILITY

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

SERIALIZABILITY

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



SERIALIZABILITY

- **View Serializable**

Any given concurrent schedule is said to be View Serializable if and only if it is View equivalent to one of the possible serial schedule.

Let *S* and *S´* be two schedules with the same set of transactions. *S* and *S´* are **view equivalent** if the following three conditions are met, for each data item *Q*,

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

CONCURRENCY CONTROL

- Process of managing simultaneous execution of transactions in a shared database, to ensure the serializability of transactions, is known as concurrency control.
- Process of managing simultaneous operations on the database without having them interfere with one another.
- Prevents interference when two or more users are accessing database simultaneously and at least one is updating data.
    - Although two transactions may be correct in themselves, interleaving of operations may produce an incorrect result.
    - Simultaneous execution of transactions over a shared database can create several data integrity and consistency problems.
    - lost updated problem
    - Temporary updated problem
    - Incorrect summery problem
- **Lost updated problem**
    - This problem occurs when two transactions that access the same database items have their operations interleaved in a way that makes the value of some database item incorrect.
    - Successfully completed update is overridden by another user.
- **Example:**
    - T1 withdraws £10 from an account with balx, initially £100.
    - T2 deposits £100 into same account.
    - Serially, final balance would be £190.
    - *Loss of T2's update!!*
    - This can be avoided by preventing T1 from reading balx until after update.

CONCURRENCY CONTROL

T1	T2
Read(X)	
X:=X-N	
	Read(X)
	X:=X+M
Write(X)	
Read(Y)	
	Write(X)
Y:=Y+N	
Write(Y)	

CONCURRENCY CONTROL

- **Temporary updated problem**
  - This problem occurs when one transaction updates a database item and then the transaction fails for some reason. The updated item is accessed by another transaction before it is changed back to its original value.
  - Occurs when one transaction can see intermediate results of another transaction before it has committed.

- Example:
  - T1 updates balx to £200 but it aborts, so balx should be back at original value of £100.
  - T2 has read new value of balx (£200) and uses value as basis of £10 reduction, giving a new balance of £190, instead of £90.
  - Problem avoided by preventing T2 from reading balx until after T1 commits or aborts.

T1	T2
Read(X) X:=X-N Write(X)	
	Read(X) X:=X-M Write(X)
Rollback Read(Y)	

CONCURRENCY CONTROL

- **Incorrect summary problem**
  - If one transaction is calculating an aggregate summary function on a number of records while other transactions are updating some of these records, the aggregate function may calculate some values before they are updated and others after they are updated.
  - Occurs when transaction reads several values but second transaction updates some of them during execution of first.

T1	T2
	Sum:=0
	Read(A)
Read(B)	Sum:=sum+A
B:=B-N	Read(B)
Write(B)	Sum:=sum+B
	.
	.
	.
	Read(X)
	Sum:=sum+X
Read(X)	
X:=X-N	
Write(X)	

CONCURRENCY CONTROL

- Example:
  - T6 is totaling balances of account x (£100), account y (£50), and account z (£25).
  - Meantime, T5 has transferred £10 from balx to balz, so T6 now has wrong result (£10 too high).
  - Problem avoided by preventing T6 from reading balx and balz until after T5 completed updates.
- **Concurrency control techniques**

Some of the main techniques used to control the concurrent execution of transaction are based on the concept of locking the data items

Conn…

Thank You…….