Paper Name: Database Management Systems

Topic: Types of Schedules

Paper Code: BCA CC410

Semester IV

By Ms. Manisha Prasad

Head, Assistant Professor,

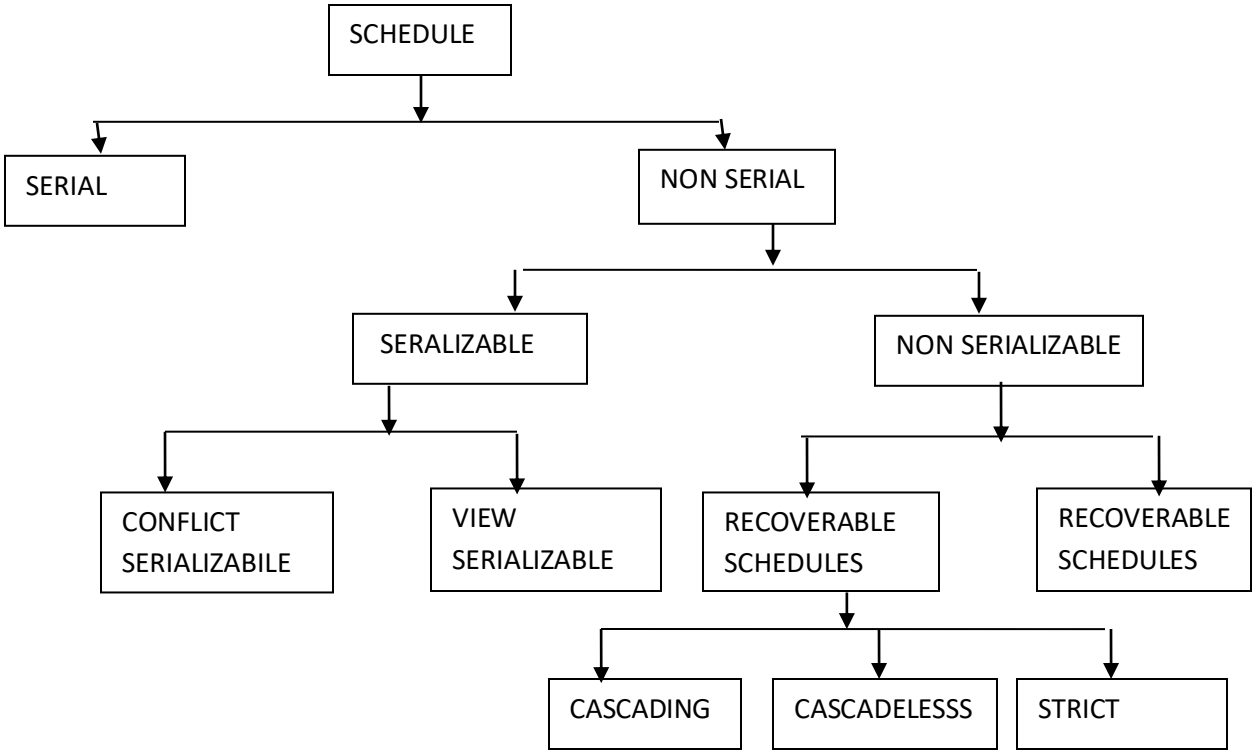
Department of Computer Science

Patna Women's College

E – mail : manisha_prasad@yahoo.com

Types of Schedules

A chronological execution sequence of a set of transactions is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks. Schedule is a process of lining the transactions and executing them. When there are multiple transactions that are running in a concurrent manner, their order of operation needs to be set so that the operations do not overlap each other. Here Scheduling is brought into play and the transactions are timed accordingly.



Serial Schedules:
Schedules in which the transactions are executed in a series , one after the other i.e. no transaction starts until a running transaction has finished execution, is called a serial schedule.

Example: Consider the following schedule involving two transactions T_1 and T_2.

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

where R(A) denotes that a read operation is performed on some data item ‘A’ and W(A) denotes write operations .This is a serial schedule since the transactions perform serially in the order T_1 —> T_2

Non-Serial Schedule:

This is a type of Scheduling where the operations of multiple transactions are interleaved. The transactions are executed in a non-serial manner, keeping the end result correct and same as the serial schedule. Unlike the serial schedule where one transaction must wait for another to complete all its operation, in the non-serial schedule, the other transaction proceeds without waiting for the previous transaction to complete. The Non-Serial Schedule can be classified as Serializable and Non-Serializable schedules.

a. **Serializable:**

This concept is used to maintain the consistency of the database. It is mainly used in the Non-Serial scheduling to verify whether the scheduling will lead to any inconsistency or not. On the other hand, a serial schedule does not need the serializability because it executes any transaction only when the previous transaction is complete. The non-serial schedule is said to be in a serializable schedule only when it is equivalent to a serial schedule, for n number of transactions. Since concurrency is allowed in this case thus, multiple transactions can execute concurrently. A serializable schedule helps in improving both resource utilization and CPU throughput. These are of two types:

1. **Conflict Serializable:**

A Non serial schedule is called conflict serializable , if it can be transformed into a serial schedule by swapping non-conflicting operations. Two operations/ instructions are said to be in conflict if all the following conditions are satisfied :

a) They belong to different transactions
b) They operate on the same data item
c) At least one of them is a write operation

2. **View Serializable:**

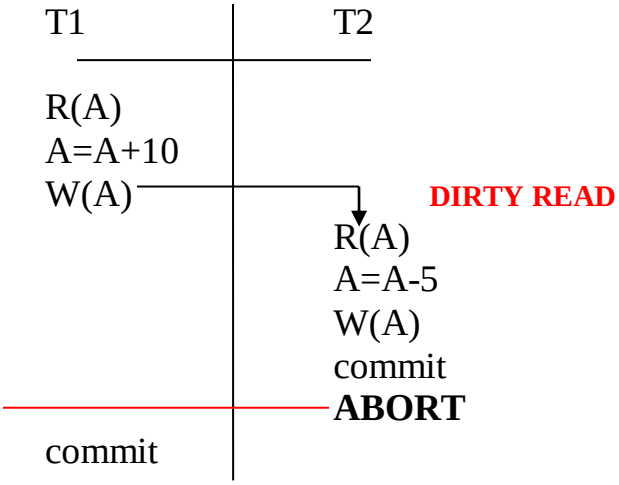
A Non Serial Schedule is called view serializable, if it is view equivalent to a serial schedule .Two schedules are said to be view equivalent, if their Initial Reads, Final Writes and Update Reads of the data items within both the schedules are same.

b. **Non-Serializable:**

The non-serializable schedule is divided into two types :-
Recoverable and Non-Recoverable Schedule.

Recoverable Schedule:
The Schedules which can handle the in between failures of the system and guarantee that they will recover the database to its consistent state are known as Recoverable Schedule.
Recoverability of a schedule is a mandatory property.

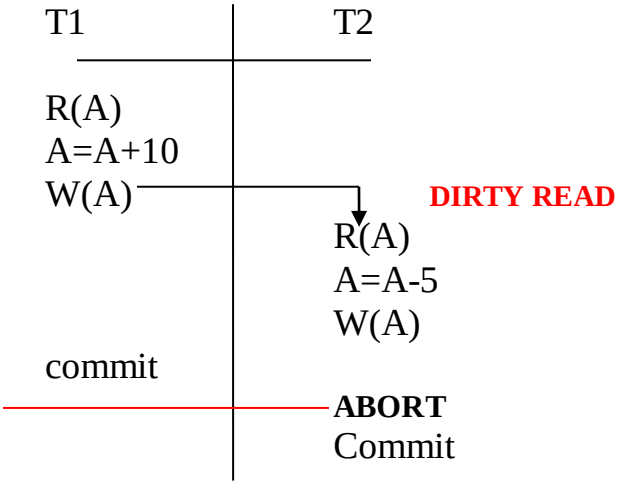
Example – Consider the following schedule involving two transactions T_1 and T_2.



As we see that it is a Serial Schedule, we know that if it executes properly it will maintain the consistency of the database.
Suppose due to some kind of hardware or software problem, some failure occurs and the schedule aborts just after committing of Transaction T2 as shown above.

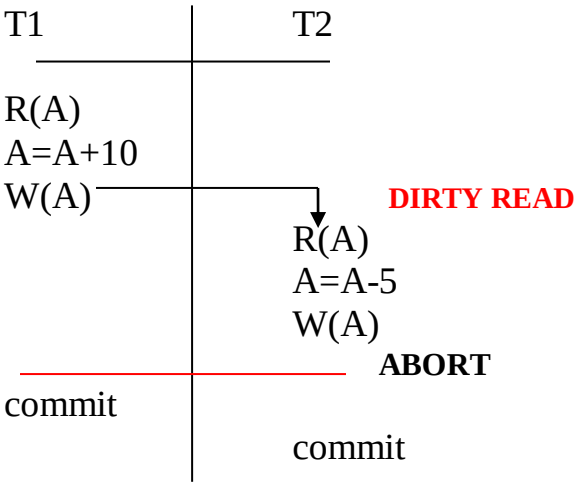
What will happen??
Transaction T1 will Rollback as it had not committed but T2 will not Rollback since it had already committed.
Will it bring back consistency in database??
Assume the original value of data item **A** in the database was **20**. T1 reads this value and adds 10 to it and writes the result 30 in the local buffer. Then T2 read the value of **A** as **30** and subtracts **5** from A and writes its value as **25** in the local buffer. When T2 performs the commit operation, the value of A is written as **25** in the database.
Now a problem occurs and the schedule aborts. Since T2 has already committed, so this transaction will not rollback, but T1 has not committed so it will rollback.
Rollback of a transaction means all the operations done by that transaction are undone.
When Transaction T1 rollbacks, it expects that the value of A should become 20 as it had read the value of A as 20 from the database and added 10 more to it making the value of A as 30. So if all its operations have reverted back the value of **A** should revert back to 20. But the case is not so, because before final commit of transaction T1 on the value of A, Transaction T2 has read the value of A, manipulated it and committed.
This example shows that even if the Schedule was Serial, some problems can lead to inconsistency in the database.
The reason for this Inconsistency was **Dirty Read** problem. This creates a dependency of T2 on T1.
When any transaction reads a temporary value from any other uncommitted transaction, this is known as Dirty Read. In the above example T2 read the uncommitted value of 'A' from T1, thus creating a dependency of T2 on T1and finally leading to the Problem of Inconsistency

CASE 1:



Now if change the order of Committing of the transaction and see the effect of Abort, we will see that there will be no inconsistency. According to the above-mentioned scenario, now T2 will rollback and T1 will not rollback since it has already committed, so the value of A remains according to the committed value of transaction T1 i.e. 30. Hence there will be no inconsistency in the database.

CASE 2:



Important points :

- **If there are no Dirty reads in the schedule, the schedule is always Recoverable.**
- **If there are Dirty Reads in the schedule, the order of committing of transactions must be in the same order in which Dirty Read has occurred. Then schedule will be Recoverable.**

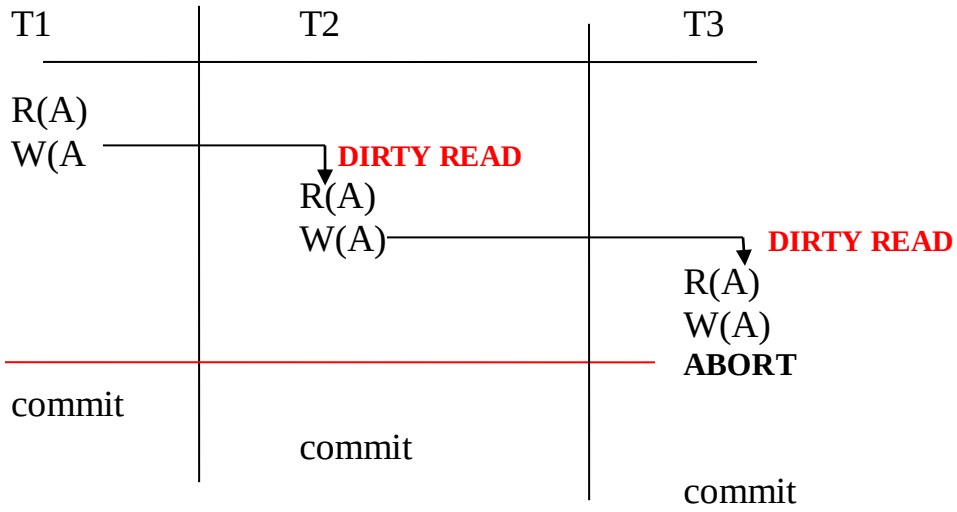
There can be three types of recoverable schedule:

Cascading Schedule: When a failure in one transaction leads to the rolling back or aborting of other dependant transactions, it is referred to as Cascading Rollback or Cascading Abort.
Example 1: Consider the following schedule involving two transactions T1 and T2.

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
abort	
	abort

It is a Recoverable schedule but it does not avoid cascading aborts. It can be seen that if T1 aborts, T2 will have to be aborted too in order to maintain the correctness of the schedule as T2 has already read the uncommitted value written by T1.

Example 2:



In the above example, if the schedule aborts, since none of the transactions have committed, so all the transactions have to roll back as T3 has read a value from T2 and T2 has read a value from T1.

Conclusion:

If there are dirty reads in a schedule, Cascading rollbacks will be there to maintain the consistency of databases irrespective of the order of commit of the participating transactions. In case of cascading rollbacks, efficiency of the system becomes low as considerable amount of work loss occurs.

Cascadeless Schedule:
Schedules in which a transaction reads a value only after all transactions whose changes it is going to read, has already committed , so that abort at any point will not trigger cascading rollbacks. Such type of schedules are called Cascadeless schedules. This method avoids aborting of a single transaction leading to a series of transaction rollbacks.
Example: Consider the following schedule involving three transactions T1, T2 and T3.

T1	T2	T3
R(A)		
W(A)		
commit		
	R(A)	
	W(A)	
	commit	
		R(A)
		W(A)
		commit

For ex :- In the above case since T1 commits a change before T2 reads the value of A, so the updated value of A has already written by T1in the database and same is the case of T3 which reads the value of A after T2 commits.
It is a strict form Recoverable schedule ensuring consistency of database. However it lowers the degree of concurrency of transaction, as one transaction waits for the other transaction to commit, whose value it wants to read

Strict Schedule:
A schedule is strict, if for any two transactions T1, T2, if a write operation of T1 precedes a conflicting operation of T2 (either read or write), then the commit or abort event of T1 also precedes that conflicting operation of T2.
In other words, T2 can read or write updated or written value of T1 only after T1 commits/aborts.

Example 1: Consider the following schedule involving two transactions T_1 and T_2.

T1	T2
R(A)	
A=A+10	
W(A)	
	W(A)
commit	
	R(A)
	commit

We see that above schedule is a serial schedule so we expect it to maintain the consistency. Moreover there is no dirty read, so we expect it to be a Cascadeless recoverable schedule. But when we closely observe, we find a **Blind Write** in the Transaction T2.
Blind Write means a transaction writes a value in a particular data item without reading its value from anywhere.

What can be the problem in such situation?????
Assume the original value of data item **A** in the database was **20**. T1 reads this value and adds **10** to it and writes the result 30 in the local buffer. Then T2 say simply writes the value of A as **100** in the local buffer. Now when T1 performs the commit operation, the value of A is written as **100** from the local buffer in the database but T1 assumes that it has updated the value of 'A' as 30. So there is some kind of inconsistency.

So to solve this problem, the approach should be as follows:-
If any transaction performs a Write operation on any data item, then no other transaction should perform a Read or Write operation on that data item before the first transaction performs the commit operation. Such schedules are known as Strict Schedules.

T1	T2
R(A)	
A=A+10	
W(A)	
commit	
	W(A)
	R(A)
	commit

This is a strict schedule since T2 reads and writes A, which is written by T1 only after the commit of T1.

It is not necessary that all strict schedules are Serial schedules. Let us look at the following example.
Example 2: Consider the following schedule involving two transactions T_1 and T_2.

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

It is a Non serial Schedule. There is no dirty read here. So it is a Cascadeless schedule. Here the transaction T2 reads the value of data item 'A' only when T1 has committed the updated value of 'A'. So it is a Strict Schedule. Hence we can say that Strict schedule is more restrictive form of Cascadeless schedule.

Non-Recoverable Schedule:

Example: Consider the following schedule involving two transactions T_1 and T_2.

T1	T2
R(A)	
W(A)	
	R(A)
	W(A)
	commit
ABORT	

Transaction T2 reads the value of data item 'A' written by T1, and commits.
T1 later aborts and rolls back, therefore the value read by T2 is wrong. Since T2 has already committed so it cannot revert back. Because T2 has proceeded with uncommitted value of A updated by transaction T1, which cannot be undone in this scenario results in Inconsistency. So this schedule is a Non-Recoverable schedule.