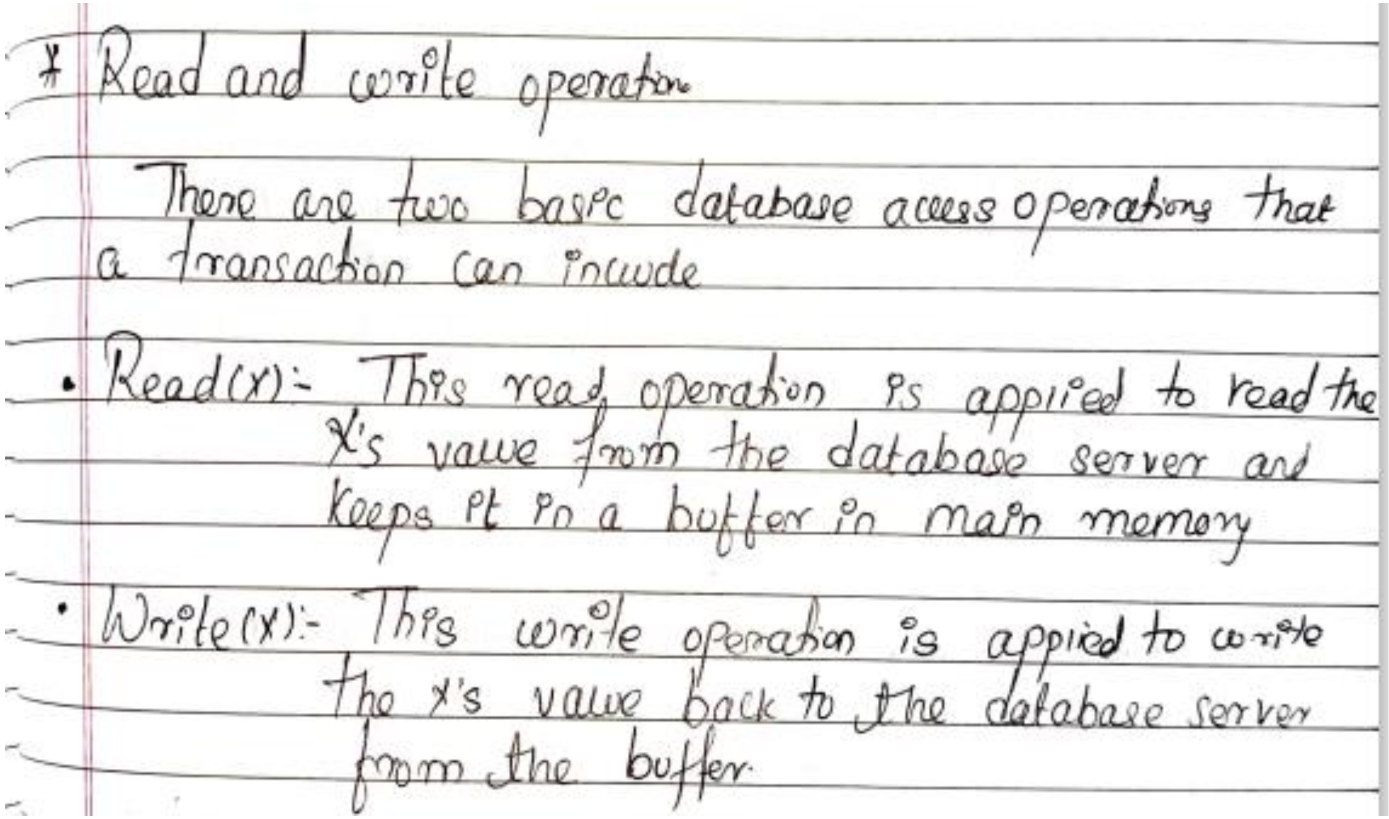
Chapter 8

Transactions

Transaction Concept

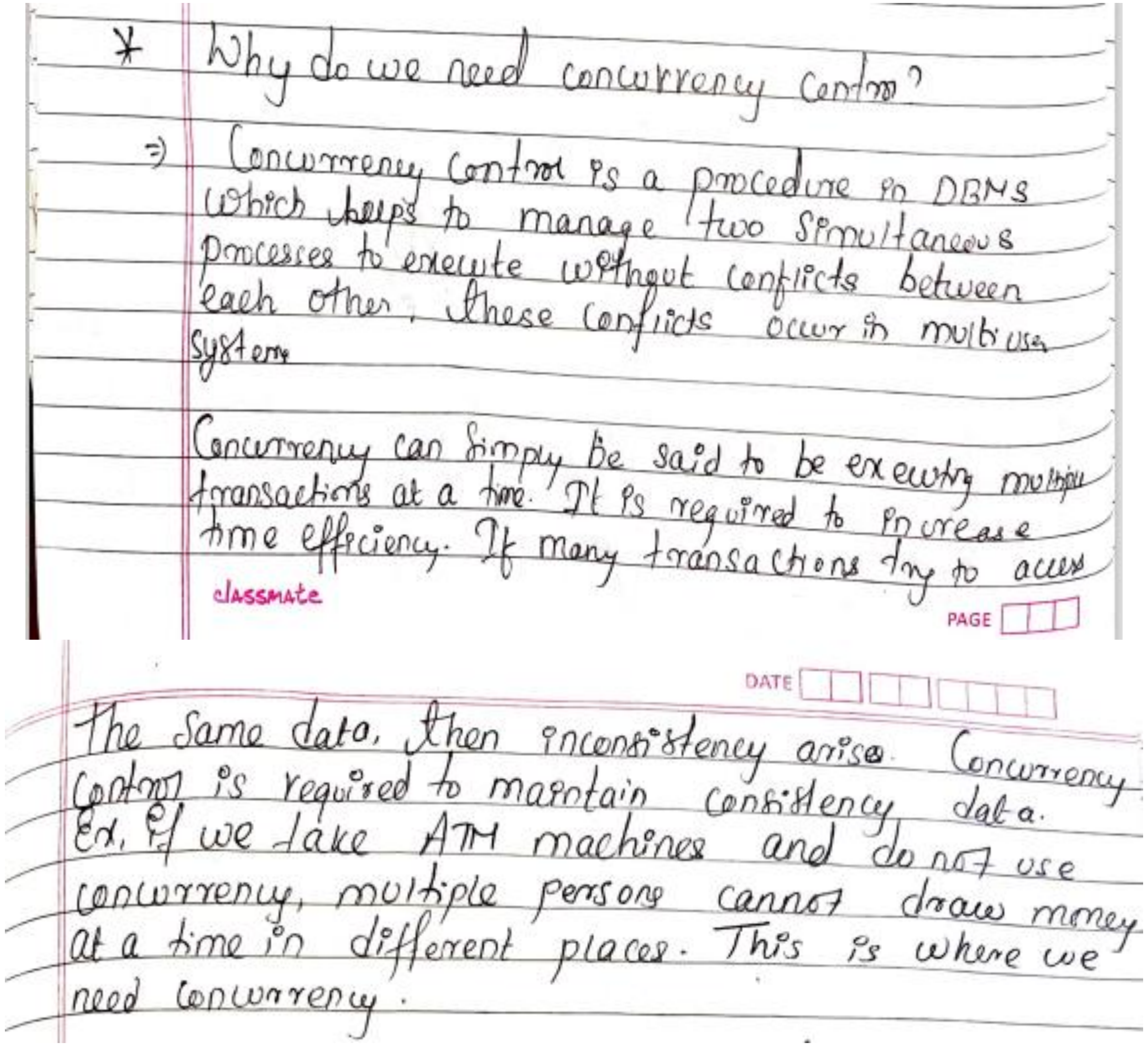
1. A ***transaction*** is a *unit* of program execution that accesses and possibly updates various data items.
2. A transaction must see a consistent database.
3. During transaction execution the database may be inconsistent.
4. When the transaction is committed, the database must be consistent.
5. Two main issues to deal with:
    - Failures of various kinds, such as hardware failures and system crashes
    - Concurrent execution of multiple transactions



* Read and write operation

There are two basic database access operations that a transaction can include

- Read(x):- This read operation is applied to read the x's value from the database server and keeps it in a buffer in main memory
- Write(x):- This write operation is appied to write the x's value back to the database server from the buffer.



* Why do we need concurrency Control?

⇒ Concurrency control is a procedure in DBMS which helps to manage two simultaneous processes to execute without conflicts between each other, these conflicts occur in multi user systems

Concurrency can simply be said to be executing multiple transactions at a time. It is required to increase time efficiency. If many transactions try to access the same data, then inconsistency arise. Concurrency control is required to maintain consistency data. Ex, if we take ATM machines and do not use concurrency, multiple persons cannot draw money at a time in different places. This is where we need concurrency.

We need concurrency control for following reasm:

- To apply isolation through mutual exclusion betwen Conflicting transaction
- To resolve read-write and write-write conflict issue
- To preserve database consistency through Constantly preserving execution obstructions.
- The system needs to control the interaction among the Concurrent transactions. This control is achieved using Concurrent-control schemes
- It helps to ensure serializability.

Problems like The Lost Update problem, the incorrect Summary problem, unrepeatable read problem will occur if we do not control concurrency.

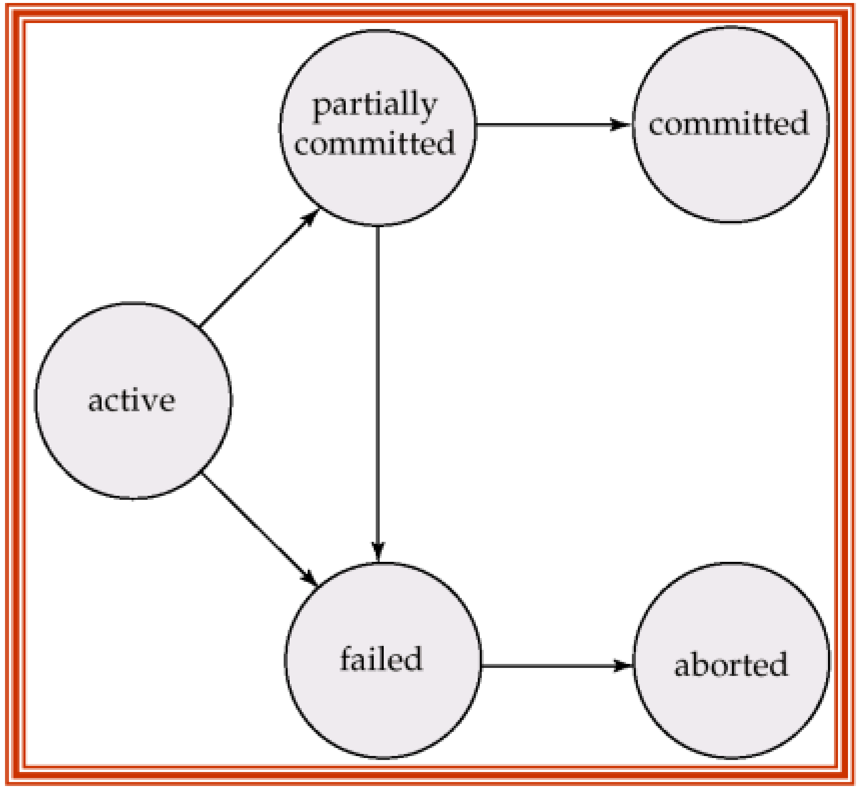
* Why do we need recovery?

=> Database systems like any other computer system are subject to failures but the data stored in it must be available when required. Some failures like system crash, system or transaction error, concurreny control enforcement, disk failure and physical problems may occur.

For this, the system must keep sufficient information to quickly recover from the failure. It must also have atomicity ie, either transaction are completed successfully and committed or the transaction should have no effect on the database. So, to prevent data lose, recovery techniques based on immediate update or backing up data can be used. The concept of transaction is fundamental to many techniques for concurrency control & recovery from failure.

Transaction State

1. **Active,** the initial state; the transaction stays in this state while it is executing
2. **Partially committed,** after the final statement has been executed.
3. **Failed,** after the discovery that normal execution can no longer proceed.
4. **Aborted,** after the transaction has been rolled back and the database restored to its state prior to the start of the transaction. Two options after it has been aborted:
   a. restart the transaction – only if no internal logical error
   b. kill the transaction
5. **Committed,** after *successful completion*.



The System Log

- **Log or Journal** : The log keeps track of all transaction operations that affect the values of database items. This information may be needed to permit recovery from transaction failures. The log is kept on disk, so it is not affected by any type of failure except for disk or catastrophic failure. In addition, the log is periodically backed up to archival storage (tape) to guard against such catastrophic failures.
- T in the following discussion refers to a unique **transaction-id** that is generated automatically by the system and is used to identify each transaction:

The System Log (cont):

Types of log record:

1. [start_transaction,T]: Records that transaction T has started execution.
2. [write_item,T,X,old_value,new_value]: Records that transaction T has changed the value of database item X from old_value to new_value.
3. [read_item,T,X]: Records that transaction T has read the value of database item X.
4. [commit,T]: Records that transaction T has completed successfully, and affirms that its effect can be committed (recorded permanently) to the database.
5. [abort,T]: Records that transaction T has been aborted.

The System Log (cont):

- protocols for recovery that avoid cascading rollbacks do not require that read operations be written to the system log, whereas other protocols require these entries for recovery.
- strict protocols require simpler write entries that do not include new_value.

Recovery using log records:

If the system crashes, we can recover to a consistent database state by examining the log and using one of the techniques described in later sections.

1. Because the log contains a record of every write operation that changes the value of some database item, it is possible to **undo** the effect of these write operations of a transaction T by tracing backward through the log and resetting all items changed by a write operation of T to their old_values.
2. We can also **redo** the effect of the write operations of a transaction T by tracing forward through the log and setting all items changed by a write operation of T (that did not get done permanently) to their new_values.

Commit Point of a Transaction:

- **Definition:** A transaction T reaches its **commit point** when all its operations that access the database have been executed successfully *and* the effect of all the transaction operations on the database has been recorded in the log. Beyond the commit point, the transaction is said to be **committed,** and its effect is assumed to be *permanently recorded* in the database. The transaction then writes an entry [commit,T] into the log.
- **Roll Back of transactions:** Needed for transactions that have a [start_transaction,T] entry into the log but no commit entry [commit,T] into the log.

Commit Point of a Transaction (cont):

- **Redoing transactions:** Transactions that have written their commit entry in the log must also have recorded all their write operations in the log; otherwise they would not be committed, so their effect on the database can be *redone* from the log entries. (Notice that the log file must be kept on disk. At the time of a system crash, only the log entries that have been *written back to disk* are considered in the recovery process because the contents of main memory may be lost.)
- **Force writing a log:** *before* a transaction reaches its commit point, any portion of the log that has not been written to the disk yet must now be written to the disk. This process is called force-writing the log file before committing a transaction.

DATE

Recovabil

Characterizing Schedules Based on Serializabi

* Concept of sChedule

When several transactions, are executing concurrently, then the order of execution of various instructions is known as schedule. They represent the Chronological order in which instructions are executed in the system

A sChedule can have many transactions in it, each comprising of a number of instructions/tasks

* Characterizing schedules based on Recoverability

It may be easy or difficult to recover transaction from system failure. The different types of schedules based on recovability are

1. Recoverable Schedule.

A schedule S is recoverable if no transaction T in S commits until all transactions T' that have written an item that T reads have committed

Here, transaction T2 is reading value written by transaction T1 and the Commit of T2 occurs after commit of T1. Hence, it is a recoverable schedule

T_1	T_2
R(X)	
W(X)	
	W(X)
Commit	R(X)
	Commit

2. Cascadeless schedule

It is the one where every transaction reads only the items that are written by committed transaction

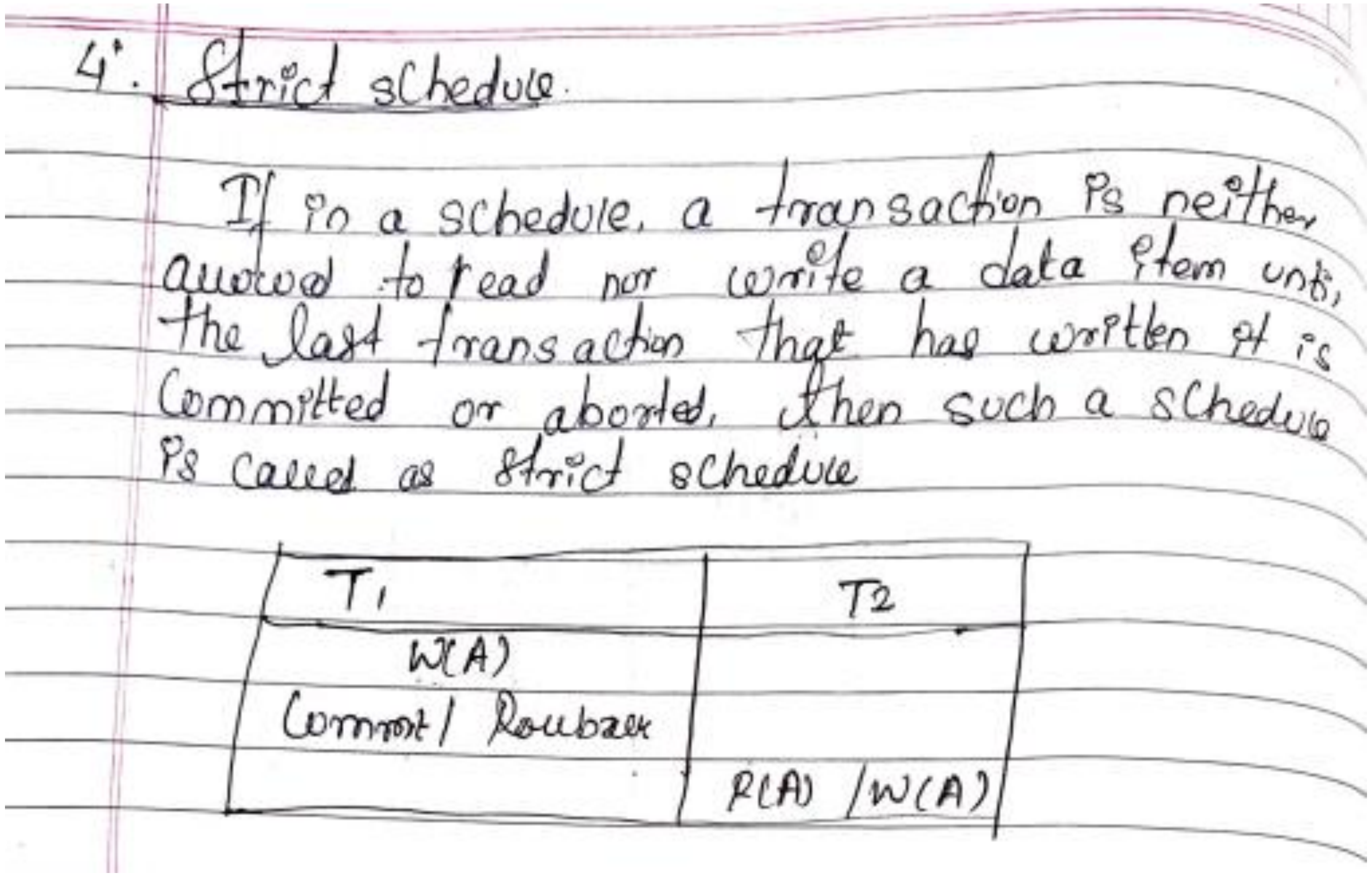
T1	T2	T3
R(X)		
W(X)		
C	R(X)	
	W(X)	
	C	R(x)
		W(x)
		C

3 Schedules requiring cascaded rollback:-

A schedule in which uncommitted transactions that read an item from a failed transaction must be rolled back. If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, then such a schedule is called as a cascading Rollback.

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

It simply leads to the wastage of CPU time.



4. Strict schedule.

If in a schedule, a transaction is neither allowed to read nor write a data item until the last transaction that has written it is committed or aborted, then such a schedule is called as strict schedule

T1	T2
W(A)	
Commit/ Rollback	
	R(A) /W(A)

2 Non-Serial sChedule:-

A schedule S is called non-serial if for any two transactions Ti and Tj participating in s, the operation of each transaction are executed non-consecutively with interleaved operations from the other transaction.

T1	T2
R(A)	
A:=A-50	
W(A)	
	R(A)
	Temp:=A*0.1
	A:A-temp
	W(A)
R(B)	
B:=B+50	
W(B)	
	R(B)
	B:=B+Temp
	W(B)

V Types of Schedules:

1. Serial Schedule

A schedule in which transactions are aligned in such a way that one transaction is executed 1st. 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 Serial schedule, as transactions are executed in a serial manner. It is always a serializable schedule.

T_1	T2
R(A)	
A:=A-50	
W(A)	
R(B)	

B:=B+50
W(B)	

	R(A)
	temp: A*0.1
	W(A)

Characterizing Schedules Based on Seriaizabi

A seriaizable schedule always leaves the database in consistent state. A concurrent execution of N transaction is called seriaizable if the execution is computationaly equivalent to a serial execution. When more than one transactions are being executed concurrently, we must have serializability in order to have same effect on the database as same serial execution does.

There are two types of schedule based on seriaizability

- Conflict Seriaizable.
- View Seriaizable.

Conflict serializable

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

Two operations are said to be conflicting if all conds satisfy:

i) Both operations should belong to 2 different transactions.

ii) They should act on same database variable.

iii) At least one of the operation should be write

- Testing for conflict serializability of a schedule:-

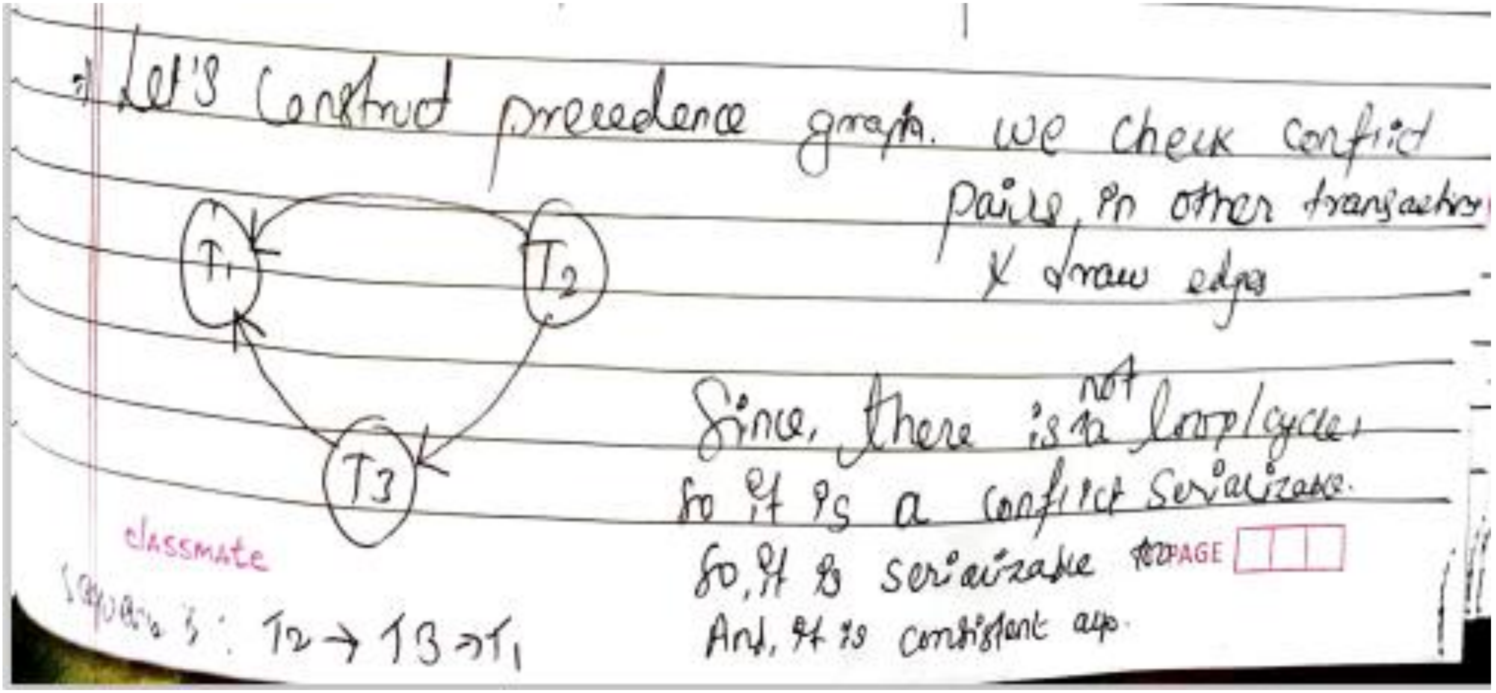
Algorithm:

- Look at only read-item (x) and write-item (x) operations
- Construct a precedence graph - a graph with directed
- An edge is created from Ti to Tj if one of the edra operation in Ti appears before a conflicting operation in Tj.
- The schedule is serializable iff the precedence graph has no cycles

Ex: Check whether given schedule is conflict serializable or not

Ex:

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



⇒ Let's construct precedence graph. we check conflict pairs in other transaction & draw edges

Since, there is ~~a~~ not loop/cycle, so it is a conflict serializable.

So, it is serializable

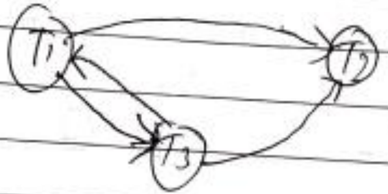
And, it is consistent also.

sequence is : T2 → T3 → T1

Ex:

T_1	T_2	T_3
R(X)		
		R(Z)
		W(Z)
	R(Y)	
R(Y)		
	W(Y)	
		W(X)
	W(Z)	
W(X)		

⇒



Since, it has cycle. So it is non-Conflict Serializable.

2 View serializable

A schedule is called view serializable if its view (equivalent) equal to a Serial Schedule (no overlapping transactions). A conflict schedule is a view serializable but if ~~the~~ the serializability contains blind write, then the view serializable is nA conflict serializable.

View Equivalence :-

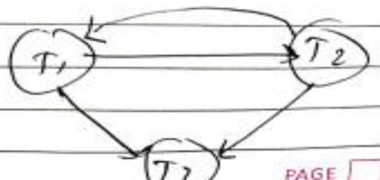
Two schedules A & B are said to be view equivalent if they follow the following condition:

- 1st read should be performed by same transaction.
- Last write should be performed by same transaction
- producer - Consumer sequence should be maintained in both S_1 and S_2: ie: T_1 (W(A)) followed by T_2 (R(A)) in S_1 and T_1 (W(A))) followed by T_2 (R(A)) in S_2

ex S_1

T_1	T_2	T_3
R(x)		
	W(x)	
	W(y)	
W(x)		
		W(x)
		R(y)

⇒ It's Precedence graph :



It Contains loop, so it is not conflict serializable. But it may be view serializable. Let's Check it further

S_1

T_1	T_2	T_3
R(x)		
	W(x)	
	W(y)	
W(x)		
		W(x)
		R(y)

S_1'

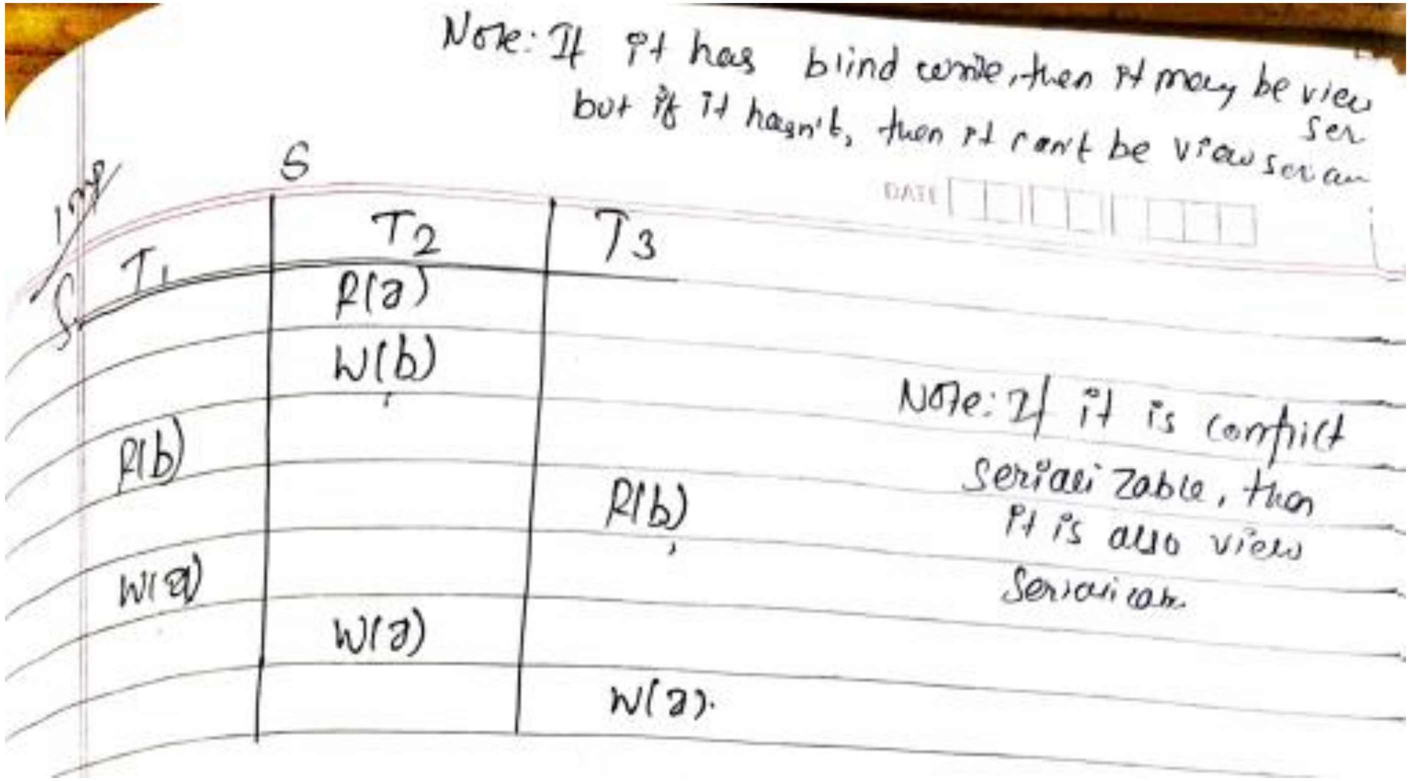
T_1	T_2	T_3
R(x)		
w(x)		
	w(x)	
	W(y)	
		w(x)
		R(y)

We draw the equivalent serial schedule of S_1 as S_1'

Here, S_1 and S_1' Satisfies all 3 view equivalent condn

- S_1 - R1(x) and S_1' - R1(x) (you can Check y also)
- S_1 - W3.(x) and S_1' - W3(x)
- S_1 - W2(y) & Consumed by S_1 - R3(y)
  S_1' - W2(y) & Consumed by S_1' - R3(y)

Thus the given S_1 is view serializable



Note: If it has blind write, then it may be view ser but if it hasn't, then it can't be view serian

S

T1	T2	T3
	R(a)	
	W(b)	
R(b)		
		R(b)
W(a)		
	W(a)	
		W(a).

Note: If it is conflict serializable, then it is also view Seriaizab.

Q. Explain whether the below schedule is

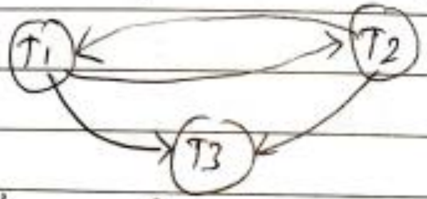
i) Conflict Serializable

ii) View serializable

iii) Conflict serializable but not view serializable

iv) View serializable but not conflict serizable.

i) Let's draw precedence graph



It has a loop, so it is not conflict Serializable

ii) Let's draw equivalent serial schedule of S as S'

S

T1	T2	T3
	R(a)	
	W(b)	
R(b)		
		R(b)
W(a)		
	W(a)	
		W(a)

S'

T1	T2	T3
	R(a)	
	W(b)	
	W(a)	
R(b)		
W(a)		
		R(b)
		W(a)

- S – $R_2(a)$ and S' – $R_2(a)$ (you can check others too)
- S – $W_3(a)$ and S' – $W_3(a)$
- S – $W_2(b)$ & Consumed by S' – $R_1(b)$
  S' – $W_2(b)$ & Consumed by S' – $R_1(b)$

Here S & S' Satisfies all 3 view equivalent condition
Hence, S is view Serializable

iii) Hence, it is not Conflict Serializable but view Serializable (as it has blind write also)

iv) It is view Serializable but not Conflict serializable because S & S' are view equivalent and the precedency graph of S has loop

Questions asked from this Chapter

Q. Define schedule and Serializability. How can you test the serializability?
(2078 - 5 marks) (2076 - 5 marks)

Q. What is Serializable schedule? How can you test a schedule for conflict serializability?
(2070 - 5 marks)

Q. Discuss ACID properties of a database transaction with suitable example. (2067 - 5 marks)
(2072 - 5 marks) (2068 - 5 marks)

Q. Describe the serial & serializable schedule? Why serializable schedule is consider correct?
(2069 - 5 marks, 2067 - 5 marks)

Q. Define the concept of recoverable, cascadeless & strict schedule, & compare them in terms of their recoverability. (2068 - 5 marks)

Q. Draw a state diagram, & discuss the typical state that a transaction goes through during transaction. (2069 - 5 marks) (2067 - 5 marks)

Q. Which of the following is conflict serializable? For each serializable schedule, draw equivalent serial schedule

i) $r_1(x)$; $r_3(x)$; $w_1(x)$; $r_2(x)$; $w_3(x)$;

ii) $r_1(x)$; $r_3(x)$; $w_3(x)$; $w_1(x)$; $r_2(x)$;

iii) $r_3(x)$; $r_2(x)$; $w_3(x)$; $r_1(x)$; $w_1(x)$;

iv) $r_3(x)$; $r_2(x)$; $r_1(x)$; $w_3(x)$; $w_1(x)$;

(2067 - 5 marks)