What is a Transaction in DBMS ?

A Transaction is a logical unit of work. It is the set of operations(basically read and write) to perform unit of work,(include small units of work). It can be viewed as a program whose execution preserves the consistency of the database. The primary goal of concurrency control and recovery scheme is to ensure that the execution of a transaction be atomic. What it means is, each transaction should access shared data without interfering with the other transactions and whenever a transaction successfully completes its execution, its effect should be fine lover, sales to complete(e.g. system failure), it should not have any effect on the student database. Transaction which successfully completes its execution is said to have been **committed**, otherwise the transaction is **aborted or rollback.** Bus if a committed transaction performance any update operation on the database, its effect must be reflected on the database even if there is a failure.

Some Points related to Transaction :

Transaction processing in a single and multi user environment

Single user : DBMS is a single user at most one user using the system. **Multi user :** DBMS is a multi-user if many users are using the system concurrently/simultaneously. For example, banking system or an Airline reservations.

Processing in multi user environment

Multiple users can access the database using the concept of multiprogramming, which allows OS to execute multiple program or processes at the same time. If a process waiting for IO transfer there is another program heading to utilise the CPU. So it is possible to share the time of the CPU for the several jobs. The process which is suspended is resumed from the point where it was suspended. Hence, concurrent execution of processes is actually interleaved which is illustrated in figure. There are two processes P1 and P2 executing concurrently in an interleaved fashion. If there are multiple CPUs, then parallel processing of multiple processes is possible which is illustrated in figure. There are two processes p3 and P4 which are executing parallely because of different CPUs.

Boundaries of a transaction :

- **Begin Transaction/ Start**
- **End Transaction / End**

We can specify explicitly the boundaries of a transaction by **begin transaction** and **end transaction** statements in an application program. A single application program may contain more than one transaction if it contains several transaction boundaries.

Types of transaction :

Read only transaction : If the database operations in a transaction do not update the database but only retrieve data. **Read write transaction :** If the database operation in a transaction retrieves as well as update the database.

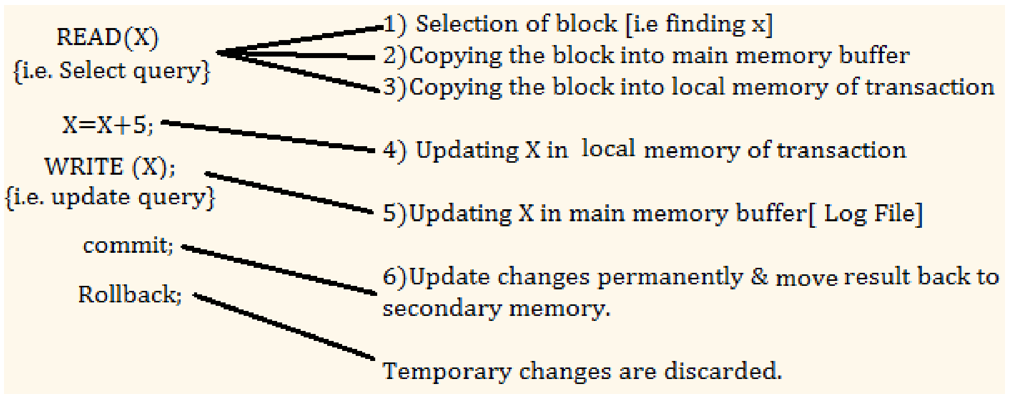
Why Recovery is Needed and The Operations of a Transaction:

Whenever a transaction is executing, either it must complete all the operations in the transaction and their changes must be permanently saved or the transaction does not have any effect on the database or any other transactions. So, for recovery purpose, the recovery manager of the DBMS needs to keep track of the following operations :

1. **Begin Transaction / Start :** It marks the beginning of transaction execution.
2. **Read (A) :** Reads a data item named A into a program variable.
3. **Write(A) :** Updation of data item (A) into database.
4. **Commit :** Transaction completed successfully.
5. **End Transaction / End :** It marks the end of transaction execution.
6. **Rollback :** It signals that the transaction has ended unsuccessfully, and so, any changes that the transaction may have applied to the database must be undone.

Steps involves in Execution of Transaction :

Database is represented as a **collection of named data items**. The data item can be a single database record or multiple database record or a field in a database record. The size of the data item is called its **granularity.**

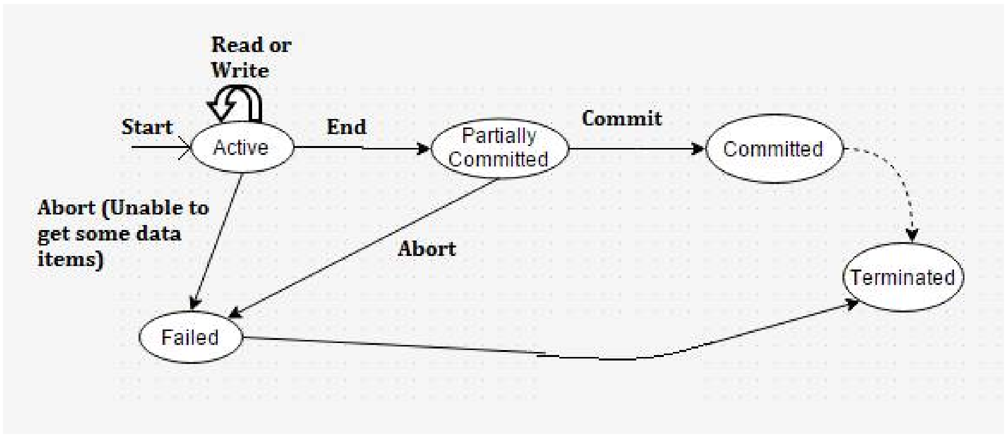

READ(X)
{i.e. Select query}
1) Selection of block [i.e finding x]
2)Copying the block into main memory buffer
3)Copying the block into local memory of transaction
X=X+5;
4) Updating X in local memory of transaction
WRITE (X);
{i.e. update query}
5)Updating X in main memory buffer[Log File]
commit;
6)Update changes permanently & move result back to secondary memory.
Rollback;
Temporary changes are discarded.

States of a transaction :

A Transaction can be in one of the following states :

1. **Active:** After the transaction has issued its start or say the transaction starts execution.
2. **Partially committed :** When the last statement is reached i.e. when the transaction ends.
3. **Aborted or rollback :** If it is found that normal execution can no longer be performed.
4. **Committed :** After successful completion.
5. **Failed :** If any failure occurs.
6. **Terminated :** Either the transaction is successfully completed or it is aborted.

Following figure shows a state transition diagram that illustrates how a transaction moves through its execution states.



Problems of Transaction and How to Solve them ??:

Transaction is associated with the following 3 problems :

- It may create an inconsistent results.
- It may create problems in concurrent execution.
- It may create an uncertainty to decide when to make changes permanent.

How to solve the above 3 problems ?

To solve the above 3 problems, we have defined some properties for the transactions and they are called **ACID** properties.

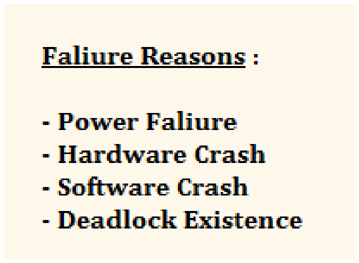
1. **A - Atomicity**
2. **C - Consistency**
3. **I - Isolation**
4. **D - Durability**

If any transaction satisfies these properties, then we can say it will be free from about problems
ACID Properties :

Transaction have 4 main properties known as ACID Properties which are explained below:

Atomicity:Atomicity refers all or none .i.e., execute all operations of transaction or none of them. So, the transactions must be atomic.

If the transaction fails due to any of the failure reason, i.e. for example, in transfer amount from account 1 to account 2, if amount is deducted from account 1 and failure occurs before the updation of account 2, then the transaction should be undo or a rollback is performed.i.e. transaction is kept back to previous state. This is done in order to avoid inconsistency.

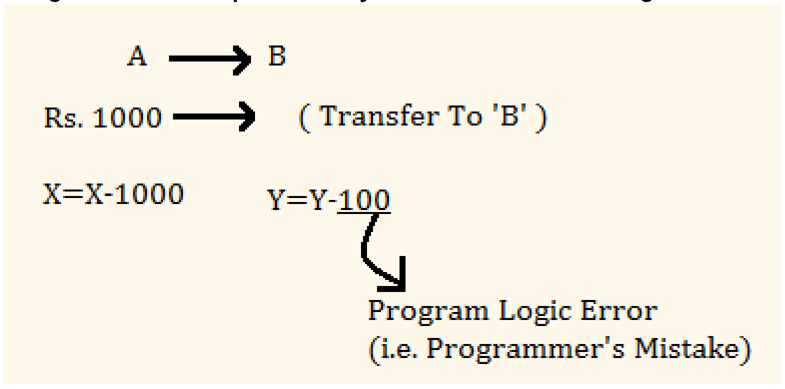

Faliure Reasons :

- **Power Faliure**
- **Hardware Crash**
- **Software Crash**
- **Deadlock Existence**

This roll back is performed by recovery management component.
For rollback, we need to know the previous state(original value) and for this, the log file is created when the transaction begins. This file maintain the initial records until the transaction is committed and is stored in the disc.
As soon as the transaction is committed or rolled back, log file is deleted.
Rollback involves undo operation , not the redo operation.

Consistency :

Programmers responsibility to take care the logical failures, not the system failures.



For example, the operations of the transactions must using Database from one consistent state to another consistent state such that it should be consistent before and after the transaction.

The consistency cannot be maintained automatically all the time. For example, if the disc crushers, then the log file gets lost. In this case we have to manually maintain the consistency.

To ensure data consistency we will use integrity constraints and they are depending on policies of organisation .

For example, the consistency criteria changes from scenario to scenario. If we have two accounts A and B, then the transaction operations involves transferring of balance between A and B. Then the consistency criteria is :

A + B = A + B

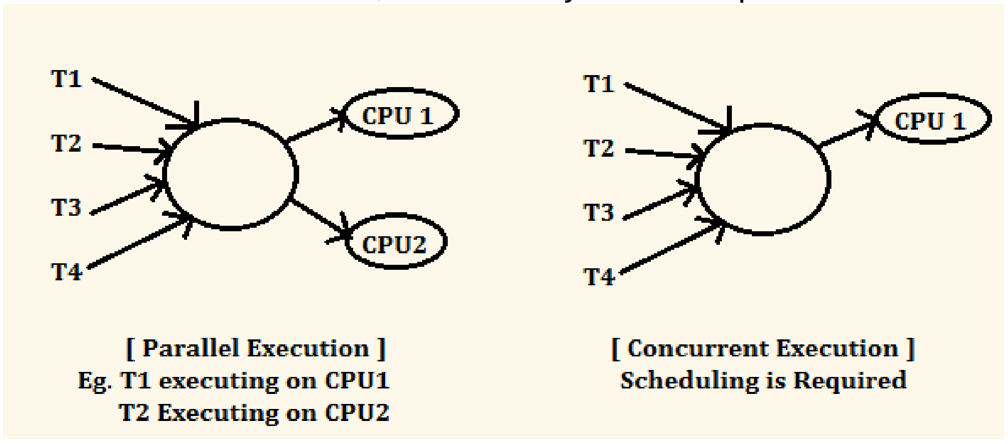
[Before Transaction] [After Transaction]

Consistency is said to be failed if atomicity fails.

Isolation:

Isolation is the property which guarantees that the execution of one transaction should not interfere with the execution of another transaction.

To ensure the isolation, concurrency control protocols must be implemented.

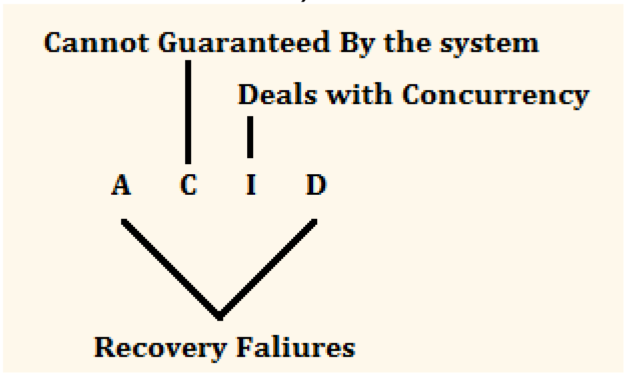


Durability :

Once a transaction commit, its updates survived, even if there is a subsequent system crash.
If we perform the commit statement, but during the transfer of records(updated) from main memory to secondary memory, failure occurs (such as power failure), and the user may feel that the transaction is committed successfully but actually it is not. In this case, the DBMS will redo the transaction.
So the database should be able to recovery under any type of crash.

Durability is hardware dependent.

To achieve durability- **RAID(Redundant Array of Independent Discs)** component is used.
Transaction management deals with Recovery(From Failure) Concurrency(From Concurrent Execution)



Implementation of Transaction in SQL

Transaction is a logical unit of work and is guaranteed to be atomic. A single SQL statement is always considered to be atomic either it complete execution without an error or it fails and leave the database unchanged. There is no explicit begin transaction statement with in SQL. The transaction initiation is done implicitly when particular SQL statements are encountered. A transaction must have an explicit end statement, which is either a commit or rollback.

Characteristics of Transaction :

The transaction characteristics are specified by a set transaction statement in SQL. The various characteristics of the transaction are :

- Access mode
- Diagnostic area size
- Isolation level

Access mode :

There are two types of access mode read only and read write. The default is read write unless the isolation read uncommitted is specified.

1. **Read only :** Read only access mode is simply used for data retrieval.
2. **Read write :** Read write access mode allows select, update, insert, delete and create command to be executed.

Diagnostic area size :

It determine the number of error conditions that can be recorded on the user on the most recently executed SQL statement. The 'n' is a parameter taking an integer value which indicates the number of conditions that can be held simultaneously in the Diagnostic area. This condition supply feedback information (error or exceptions) to the user or program on the 'n'.

Isolation level :

It controls the extent to which a transaction is exposed to the actions of other transactions executing concurrently. It is specified using the statement ISOLATION LEVEL . The value of can be

1. Read uncommitted
2. Read committed
3. Repeatable read
4. Serializable

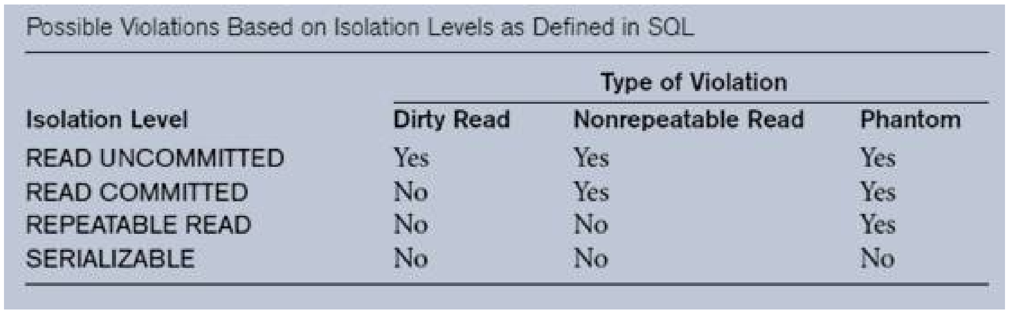
The default isolation for some system is serializable and for some systems is read committed. The serializable does not allow violations that cause dirty read, unrepeatable read, phantoms.

Dirty read :
A transaction T1 may read the update of a transaction T2, which has not yet completed. If T2 fails and is aborted then T1 would have read a value that does not exist and is incorrect.

Non repeatable read :
A transaction T1 may read a given value from a Table. If another transaction T2 letter update that value and T1 reads that value again, T1 will see a different value.

Phantoms :
Transaction T1 may read a set of rows from a Table, perhaps based on some condition specified in the SQL WHERE clause. Now suppose that a transaction T2 insert a new row that also satisfies the WHERE clause condition used in T1, into the table used by T1. If T1 is repeated, then T1 will see a Phantom, a row that previously did not exist.

Possible Violations Based on Isolation Levels as Defined in SQL :



Possible Violations Based on Isolation Levels as Defined in SQL

	Type of Violation		
Isolation Level	**Dirty Read**	**Nonrepeatable Read**	**Phantom**
READ UNCOMMITTED	Yes	Yes	Yes
READ COMMITTED	No	Yes	Yes
REPEATABLE READ	No	No	Yes
SERIALIZABLE	No	No	No

Serial and Concurrent Schedule

What is a Schedule ?

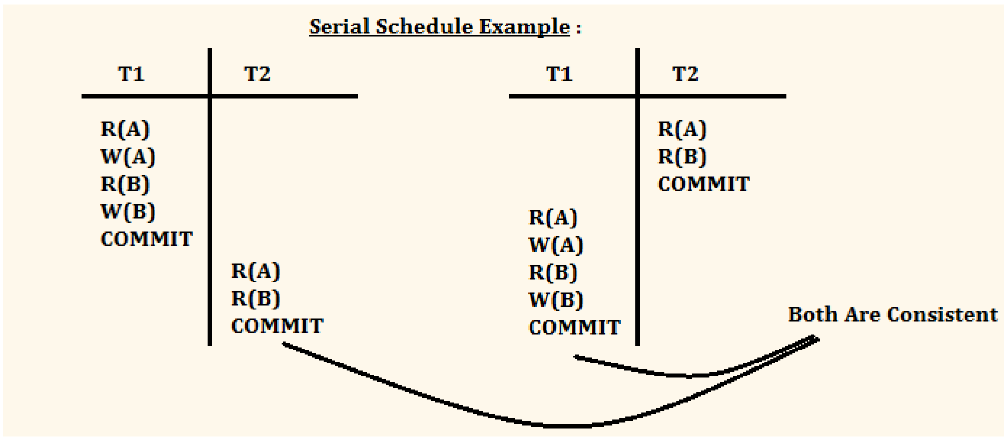
Definition 1: The order of the execution of the transactions is known as schedule.
Definition 2 : The time order sequence of operations among transactions.

Two types of schedules

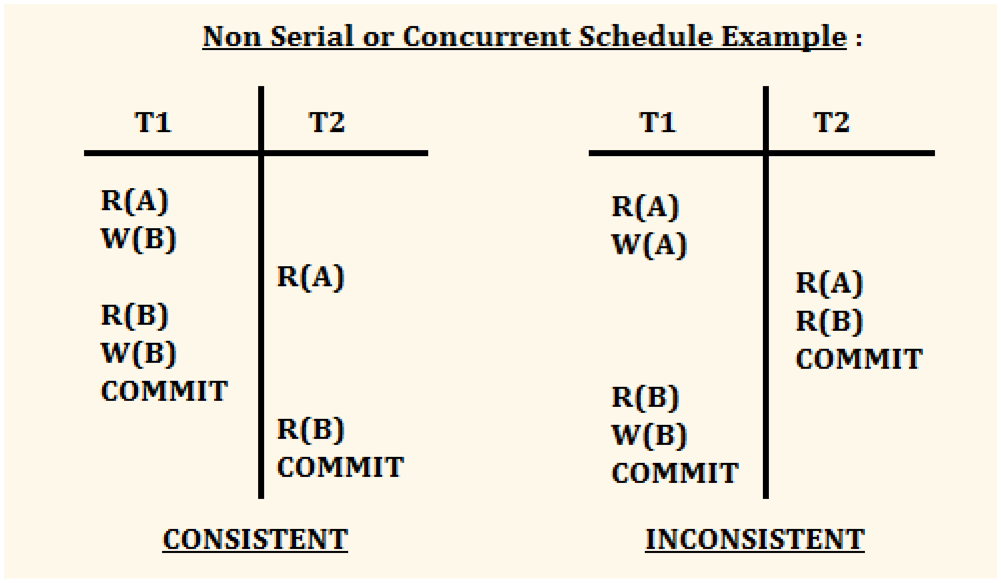
- Serial schedule
- Non serial schedule or concurrent schedule

Serial Schedule	**Concurrent Schedule**
The transactions are completed one after another. For example, if there are 3 transactions T1, T2, T3, then, Firstly, T1 complete its execution. Then after the commit of T1 transaction, transaction T2 completes, and in the last T3 completes.	Interleaved or simultaneous execution of two or more transaction is called concurrent schedule. In Concurrent Schedule, CPU time is being shared by different transactions.
Advantage : Every schedule is a serial schedule, So no problems of Isolation occurs.	**Advantage : 1.** If T1 needs DMA processor, CPU can execute T2. Hence Throughput increases. 2. Better Resource Utilization.3. Less Response Time.
Drawback : 1. THROUGHPUT of Transaction is less, since the CPU is idle if transaction needs IO. **Solution :** Use Concurrent Schedule.	**Drawback :** Problems of Isolation must be managed.

Example of serial schedule :



Example of Non Serial or Concurrent Schedule :



Non Serial or Concurrent Schedule Example :

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

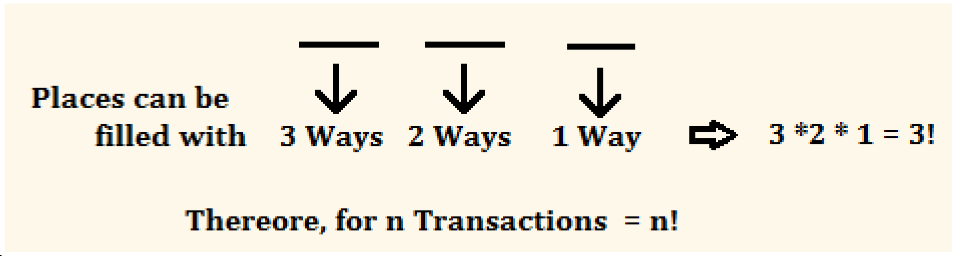
CONSISTENT

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

INCONSISTENT

Some Questions on Serial Schedule and Concurrent Schedule :

Question 1 :
How many number of serial schedules can be formed with and transactions?
Solution : let us consider three transaction T1 T2 and T3.



Places can be filled with 3 Ways 2 Ways 1 Way ⇨ 3 *2 * 1 = 3!

Thereore, for n Transactions = n!

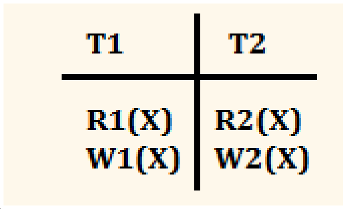
Question 2 :

How many concurrent schedules can be formed using transaction T1 and T2 on n1 n2 operations respectively?

Solution :

Consider n1 = 2, and n2 = 2

For two transactions T1 and T2 :



T1	T2

R1(X)
W1(X)
	R2(X)
	W2(X)

a) R1(X) W1(X) R2(X) W2(X)
b) R1(X) R2(X) W2(X) W1(X)
c) R1(X) R2(X) W1(X) W2(X)
d) R2(X) W2(X) R1(X) W1(X)
e) R2(X) R1(X) W1(X) W2(X)
f) R2(X) R1(X) W2(X) W1(X)

Therefore 6 Schedules = (4!) / (2! 2!) = $^4C_2$

Therefore, with n1 and n2 Operations :

(n1 + n2)! / (n1! . n2!)

Question 3 :

How many concurrent schedules can be formed using 3 transactions on n1, n2, n3, entry operations respectively?

Solution :

(n1 + n2 + n3)! / (n1! . n2! . n3!)

Question 4 :

Assume there are M transactions and number of operations in the transactions are n1, n2, n3,...,n_m. How many number of non serial schedules can be formed?

Solution :

The number of non Serial Schedules formed will be :

(n1 + n2 + n3 + ... + n_m)! / (n1! . n2! . n3! .... n_m!)

Question 5 :

Assume there are two transactions T1 and T2. T1 contains 2 operations T2 and to contains 5 operations. Find the number of serial and non serial schedules?

Solution :
Number of Serial Schedules = 2
Number on Concurrent/Nonserial Schedules = (2+5)! / (2! . 5!) = 21

Question 6 :

Let there are three transactions T1, T2, T3. T1 contains M operations, T2 contains N operations, T3 contains P operations. Find the total schedule possible?

Solution :
(m + n + p)! / (m! . n! . p!)

We can also write as :

${}^{(m+n+p)}C_m * {}^{(n+p)}C_n * {}^{p}C_p$

Serializability in Database

A schedule S of n transactions is serializable if it is equivalent to some serial schedule of the 'n' transactions. Every serializable schedule is consistent i.e. it is not suffering from RW, WR, WW etc. The concept of serializability of schedules is used to identify which schedules are correct when transaction executions have interleaving of their operations in the schedules. Serializable schedules are always considered to be correct when concurrent transactions are executing.

Difference between serial schedule and a serializable schedule

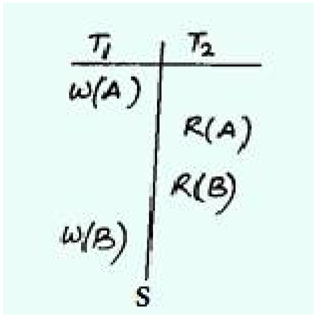
1. The main difference between the serial schedule and the serializable schedule is that in serial schedule, no concurrency is allowed whereas in serializable schedule, concurrency is allowed.
2. In serial schedule, if there are two transaction executing at the same time and if no interleaving of operations is permitted, then there are only two possible outcomes :
    - Execute all the operations of transaction T1 (in sequence) followed by all the operations of transaction T2 (in sequence).
    - Execute all the operations of transaction T2 (in sequence) followed by all the operations of transaction T1 (in sequence).

   In Serializable Schedule, if there are two transaction executing at the same time and if interleaving of operations is allowed, there will be many possible orders in which the system can execute the individual operations of the transactions.

3. In serializable schedule, the concurrent execution of schedule should be equal to any serial schedule so that schedules are always considered to be correct, when transaction executions have interleaving of their operations in the schedules.

Example of Serializable Schedule

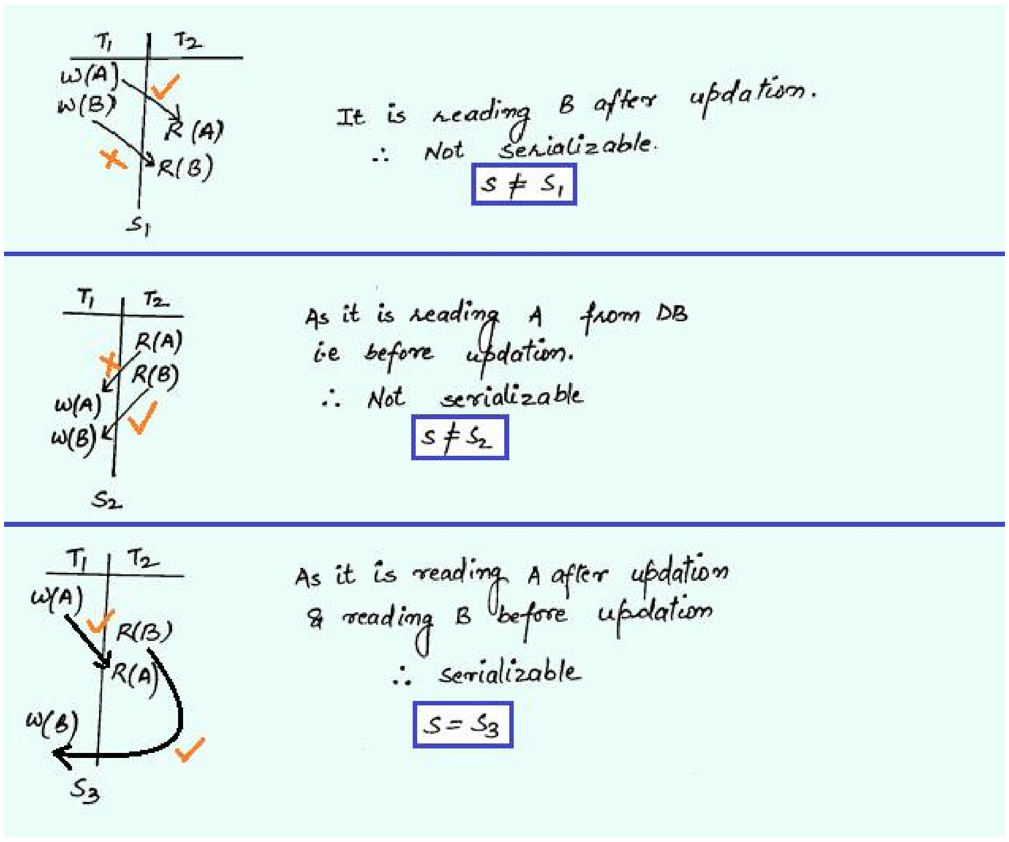
Let us consider a schedule S.



What the schedule S says ??

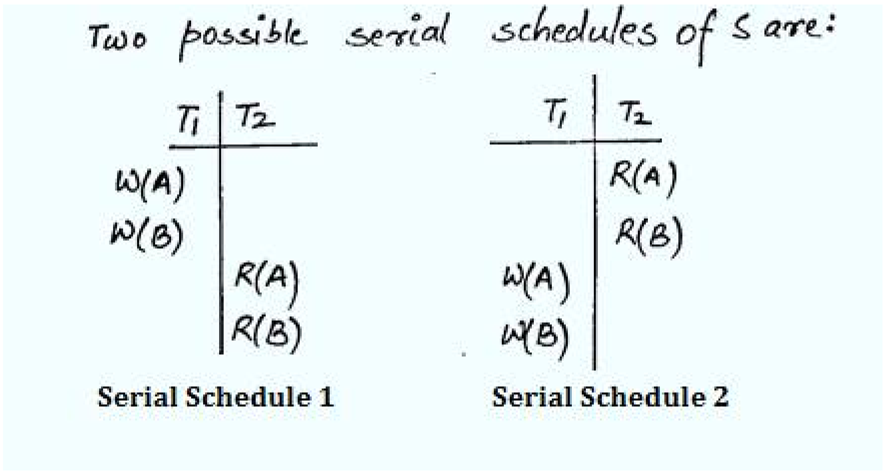
- Read A after updation.
- Read B before updation.

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

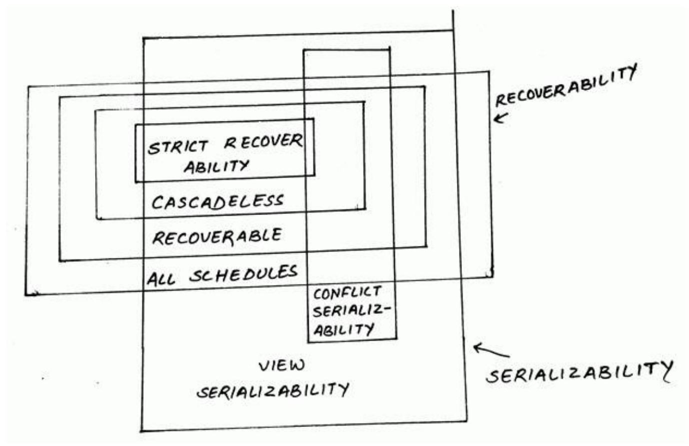


Example of Serial Schedule :

Consider the above schedule S. The serial schedules will be



Relation Between Serializability and Recoverability :



When are 2 schedules equivalent?

There are three types of equivalence of schedules :

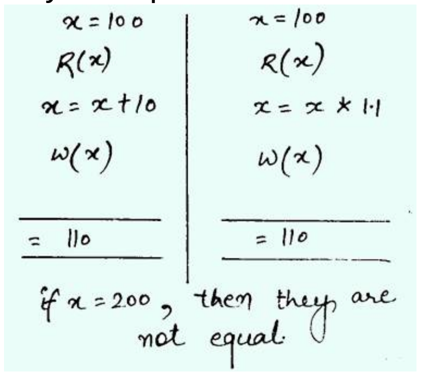
- Result equivalence
- Conflict equivalence
- View equivalence

Based on the types of equivalence, we define the **types of serializability**. There are accordingly three types of serializability which are:

- Results serializable
- Conflict serializable
- View serializable

Result Equivalence and Result Serializable :

In results equivalence, the end result of schedules heavily depend on input of schedules. The final values are calculated from both schedules (given and serial) and check whether they are equal. Result Serializable are not generally used because of lengthy process.



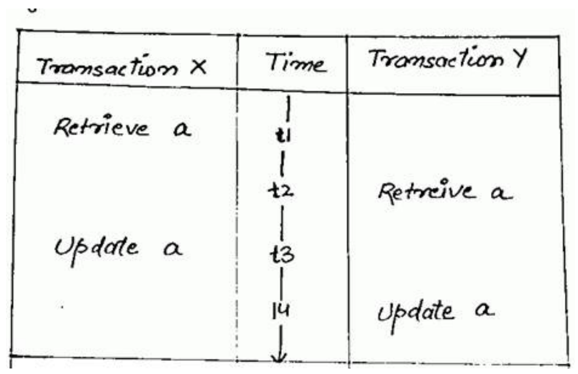
$x = 100$	$x = 100$
$R(x)$	$R(x)$
$x = x + 10$	$x = x * 1.1$
$W(x)$	$W(x)$
$= 110$	$= 110$

if $x = 200$, then they are not equal.

Concurrency Problems in Transaction

Several problems can occur when concurrent transactions execute in an uncontrolled manner. Some Concurrency Problems in transaction are :
The Lost Update Problem
The uncommitted dependency problem/The Temporary Update (or Dirty Read) Problem
The inconsistent analysis problem/The Incorrect Summary Problem
The Unrepeatable Read Problem
The Lost update problem

Consider the following example :

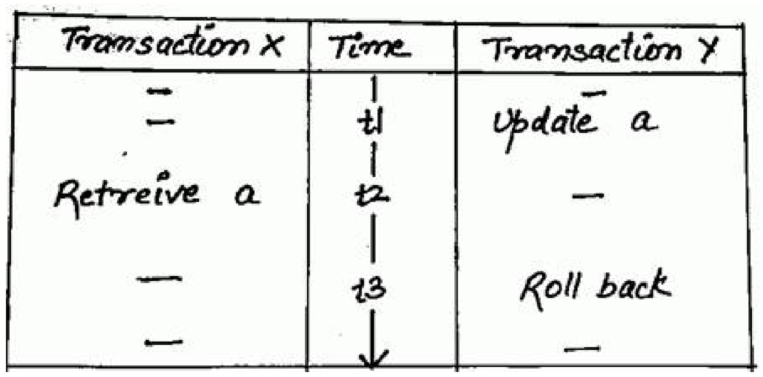


Transaction X	Time	Transaction Y
Retrieve a	t1	
	t2	Retrieve a
Update a	t3	
	t4	Update a

In this example

Transaction A retrieves some tuple X at time T1.
Transaction B retrieves same tuple X at time T2.
Transaction A update the tuple X at time T3.
Transaction B update the same tuple at time T4 on the base of the values available at time T2.
Transaction A's update is lost at time T4, since transaction B overwrites it.
The uncommitted dependency problem

Consider the following example :

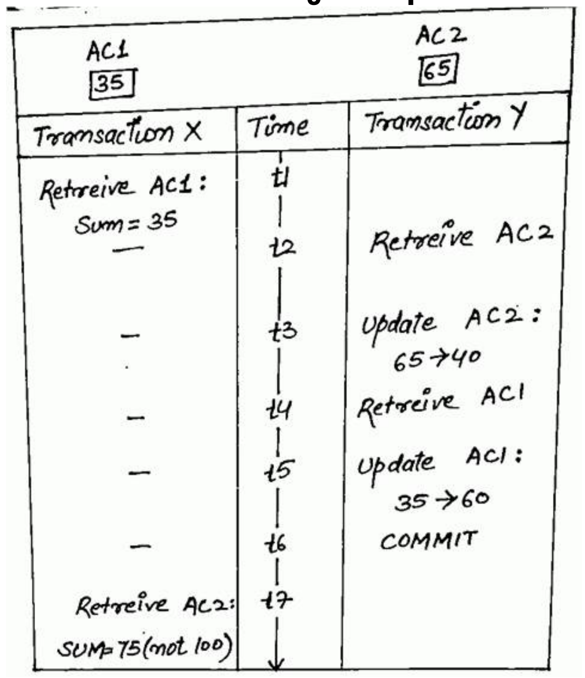


Transaction X	Time	Transaction Y
—		—
—	t1	Update a
Retreive a	t2	—
—	t3	Roll back
—		—

In this example

Transaction B update the tuple at time T1.
Transaction A retrieves the same tuple X and use the updated results at time T2.
Update (made by transaction be at time T1) is then update at time T3.
The inconsistent analysis problem

Consider the following example :



AC1 $\boxed{35}$ AC2 $\boxed{65}$

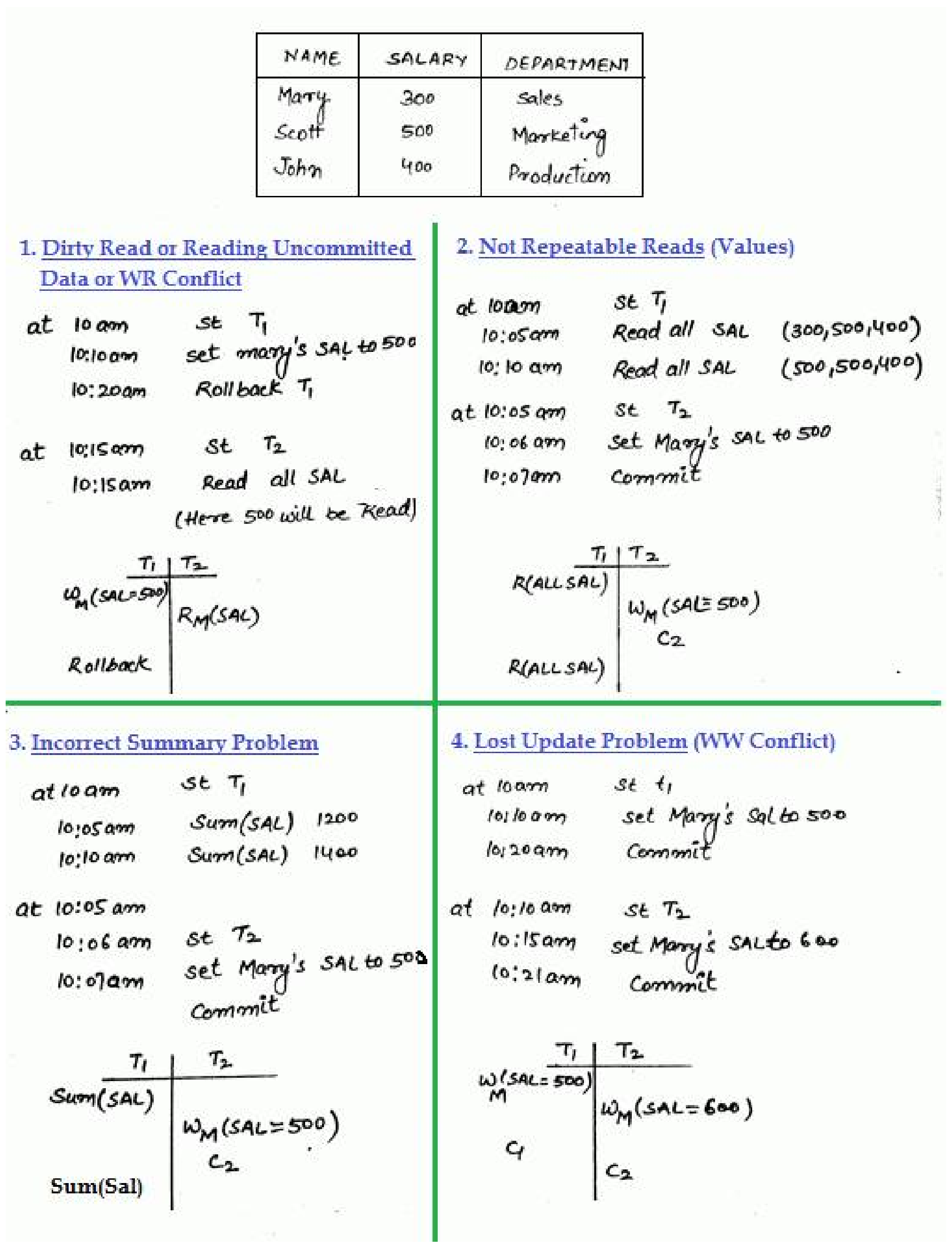
Transaction X	Time	Transaction Y
Retreive AC1: Sum = 35	t1	
—	t2	Retreive AC2
—	t3	Update AC2: 65→40
—	t4	Retreive AC1
—	t5	Update AC1: 35→60
—	t6	COMMIT
Retreive AC2: SUM=75 (not 100)	t7	

In this example, transaction A is used to find the sum of account balances and transaction B is used to transfer an amount 20 from account 2 to account 1.
At time T1, transaction A retrives AC 1 and some becomes 50.

Transaction B retrives AC 2 at time T2. And update it at time T3 as a result balance of AC 2 becomes 90.
Transaction retrives AC 1 at time T4 and update balance of AC 1 becomes 70.
Transaction B commits at time T6.
At time T7, transaction A retrieve AC 2 and sum becomes 140 but not 160. The result produced by A is wrong. Hence we say that transaction A has seen an inconsistent state.

The Unrepeatable Read Problem
Another problem that may occur is called unrepeatable read, where a transaction T reads the same item twice and the item is changed by another transaction T between the two reads. Hence, T receives different values for its two reads of the same item.

Another Example : Describing all the Concurrency Problems (Including Phantom Row and Unstable Errors) :



NAME	SALARY	DEPARTMENT
Mary	300	Sales
Scott	500	Marketing
John	400	Production

1. Dirty Read or Reading Uncommitted Data or WR Conflict

at	10 am	st T_1
	10:10am	set mary's SAL to 500
	10:20am	Rollback T_1
at	10:15am	st T_2
	10:15am	Read all SAL
		(Here 500 will be Read)

T_1	T_2
W_M(SAL=500)	
	R_M(SAL)
Rollback	

2. Not Repeatable Reads (Values)

at 10am	st T_1	
10:05am	Read all SAL	(300,500,400)
10:10 am	Read all SAL	(500,500,400)
at 10:05 am	st T_2	
10:06 am	Set Mary's SAL to 500	
10:07am	Commit	

T_1	T_2
R(ALL SAL)	
	W_M (SAL= 500)
	C_2
R(ALL SAL)	

3. Incorrect Summary Problem

at 10am	st T_1	
10:05am	Sum(SAL)	1200
10:10am	Sum(SAL)	1400
at 10:05 am		
10:06 am	st T_2	
10:07am	set Mary's SAL to 500	
	Commit	

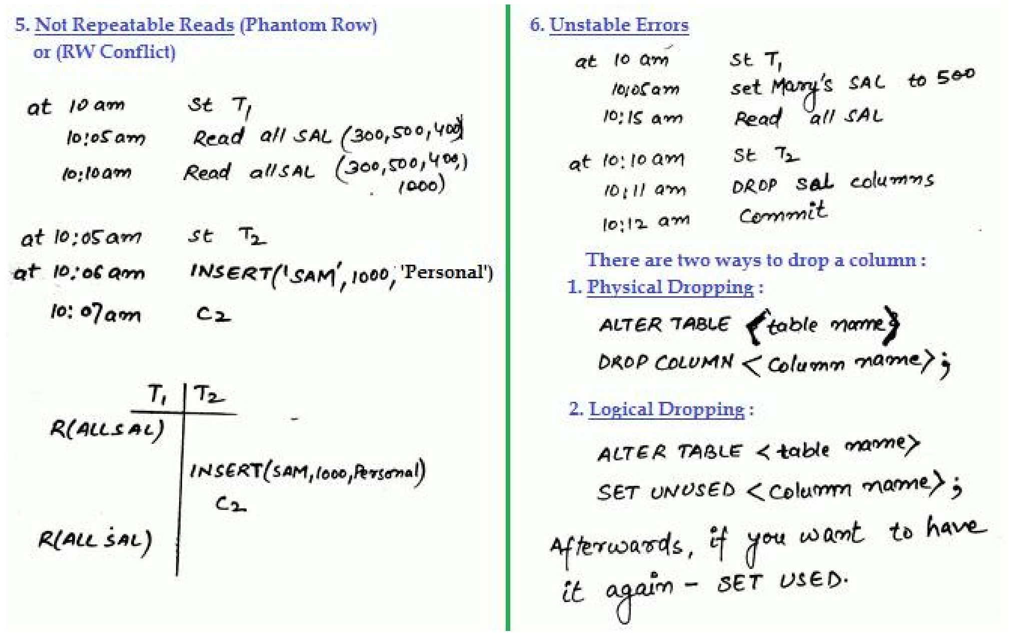
T_1	T_2
Sum(SAL)	
	W_M(SAL=500)
	C_2
Sum(Sal)	

4. Lost Update Problem (WW Conflict)

at	10am	st t_1
	10:10am	set Mary's Sal to 500
	10:20am	Commit
at	10:10 am	st T_2
	10:15am	set Mary's SAL to 600
	10:21am	Commit

T_1	T_2
W_M(SAL=500)	
	W_M(SAL=600)
C_1	
	C_2

Phantom Row and Unstable Errors : Concurrency Problems in Transaction



5. Not Repeatable Reads (Phantom Row) or (RW Conflict)

at 10 am	st T_1
10:05 am	Read all SAL (300, 500, 400)
10:10 am	Read all SAL (300, 500, 400, 1000)
at 10:05 am	st T_2
at 10:06 am	INSERT('SAM', 1000, 'Personal')
10:07 am	C_2

T_1	T_2
R(ALL SAL)	
	INSERT(SAM, 1000, Personal)
	C_2
R(ALL SAL)	

6. Unstable Errors

at 10 am	St T_1
10:05 am	set Mary's SAL to 500
10:15 am	Read all SAL
at 10:10 am	St T_2
10:11 am	DROP SAL columns
10:12 am	Commit

There are two ways to drop a column :

1. Physical Dropping :

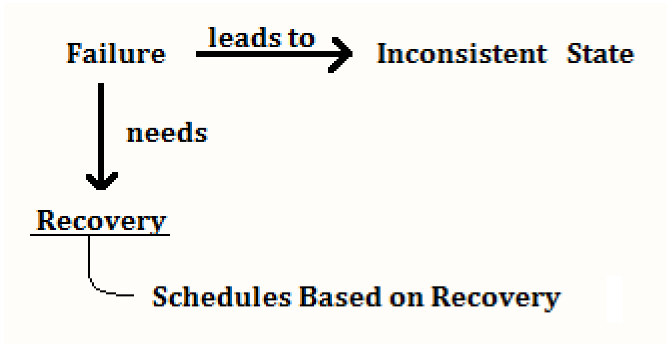
```
ALTER TABLE {table name}
DROP COLUMN <Column name>;
```

2. Logical Dropping :

```
ALTER TABLE <table name>
SET UNUSED <Column name>;
```

Afterwards, if you want to have it again - SET USED.

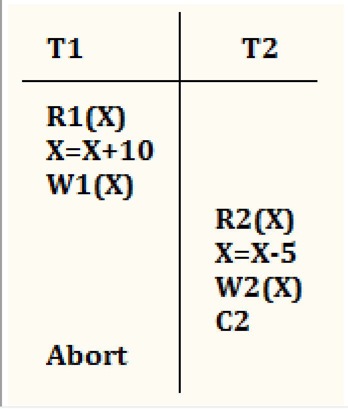
Classification of Schedule Based on Recovery



The schedules can be classified based on recovery as :

Irrecoverable schedule
Recoverable schedule
Cascading Schedule
Cascadeless Schedule
Strict Schedule : More restrictive than Cascadeless

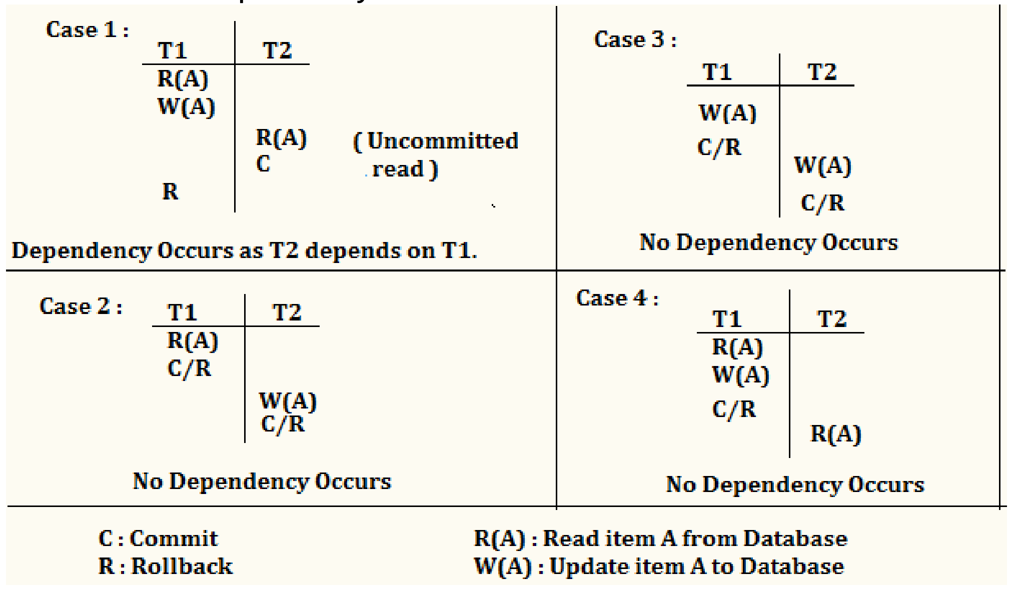
Irrecoverable schedule

Irrecoverable Schedule :	
	Let initially x=5 reads by transaction T1. X is updated by T1 to 15 which is read by T2. Now, T2 updates X again to 10 and is stored permanently by C2 (commit) statement. But, now T1 is aborted, thus while reading X, T1 will get its value as x=10 (not 5). Thus this is irrecoverable schedule as the rollback of a committed transaction is not possible. To be recoverable, the commit operation of T2 must comes after the abort operation of T1.

T1	T2
R1(X)	
X=X+10	
W1(X)	
	R2(X)
	X=X-5
	W2(X)
	C2
Abort	

Rollback is not possible in the following cases :

If transaction is committed.
If the disc crashes. Therefore the log file get lost.
Cases: Where Dependency Occurred ??



Case 1 :

T1	T2
R(A)	
W(A)	
	R(A)
	C
R	

(Uncommitted read)

Dependency Occurs as T2 depends on T1.

Case 3 :

T1	T2
W(A)	
C/R	
	W(A)
	C/R

No Dependency Occurs

Case 2 :

T1	T2
R(A)	
C/R	
	W(A)
	C/R

No Dependency Occurs

Case 4 :

T1	T2
R(A)	
W(A)	
C/R	
	R(A)

No Dependency Occurs

C : Commit
R : Rollback
R(A) : Read item A from Database
W(A) : Update item A to Database

An uncommitted read only is known as **dependency. Uncommitted read means** that the value updated by one transaction is read by another transaction before its commit or rollback.

Dependency occurs in a schedule in which a transaction reads a data item previously updated by another transaction, such that the read is done before commit of another one. i.e. in the case 1, T2 depends on T1 as the updated value of A is read by transaction T2 before any commit in transaction T1.

Case 1 is an irrecoverable Schedule because the wrong value(i.e. updated value) of A is read by T2 and cannot be rollback due to commit statement. or **Case 1 is an irrecoverable Schedule** because T2 reads a updated value before the commit operation of T1.

Case 2 and Case 3 are recoverable Schedules and no dependency occurs(WR) between T1 and T2 because the transaction is not reading any updated value.(It is updating always.) However, RW and WW dependency occurs.

Case 4 is also a recoverable schedules because commit or rollback of transaction T2 (which is reading the updated value) is to be delayed until the transaction T1 is committed or aborted.

Question : What are the precautions for a schedule to be recoverable?

or

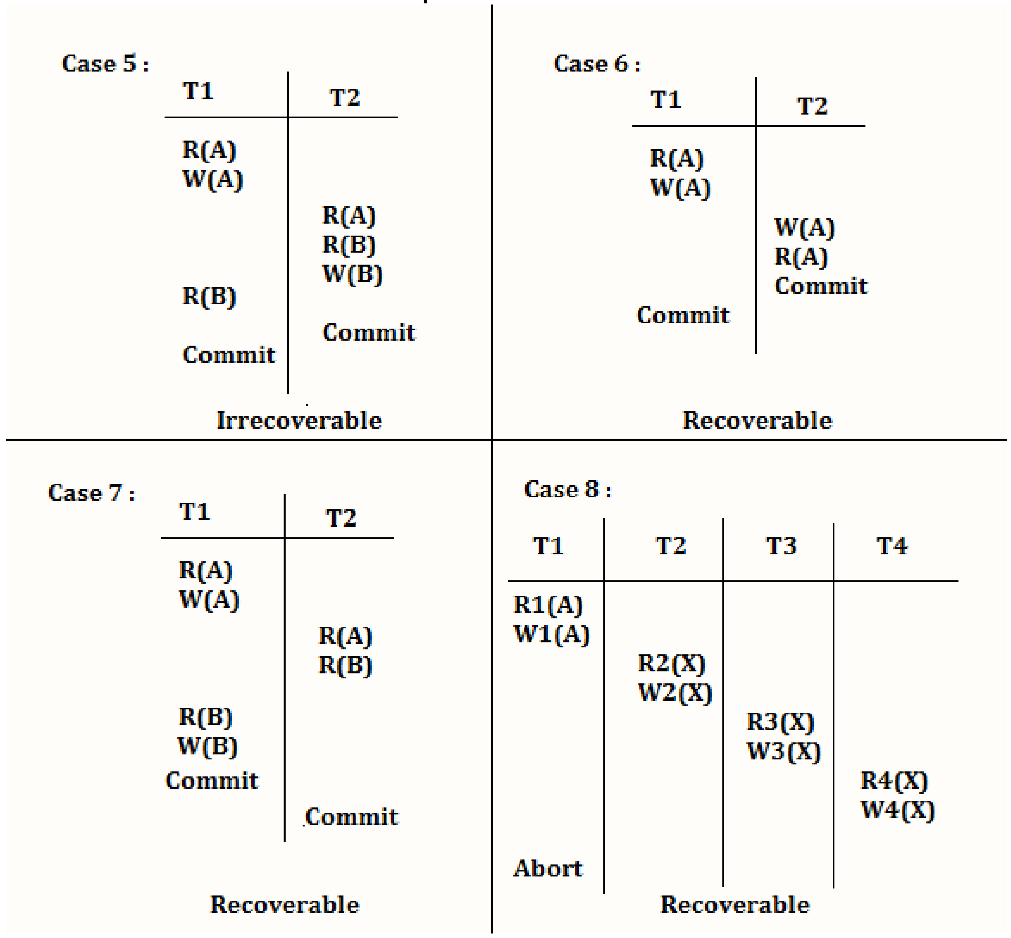
How to remove dependency if an uncommitted read occurs ?

Solution :

1. Find out the dependent transaction by finding an uncommitted read.
2. Make sure the dependent transaction should be committed later.

Recoverable schedule

A Schedule in which for every pair of transaction T_i and T_j such that T_j reads a data item previously written by T_i, the commmit operation of T_j should appear after commit or abort operation of T_i.



Case 5 :

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

Irrecoverable

Case 6 :

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

Recoverable

Case 7 :

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

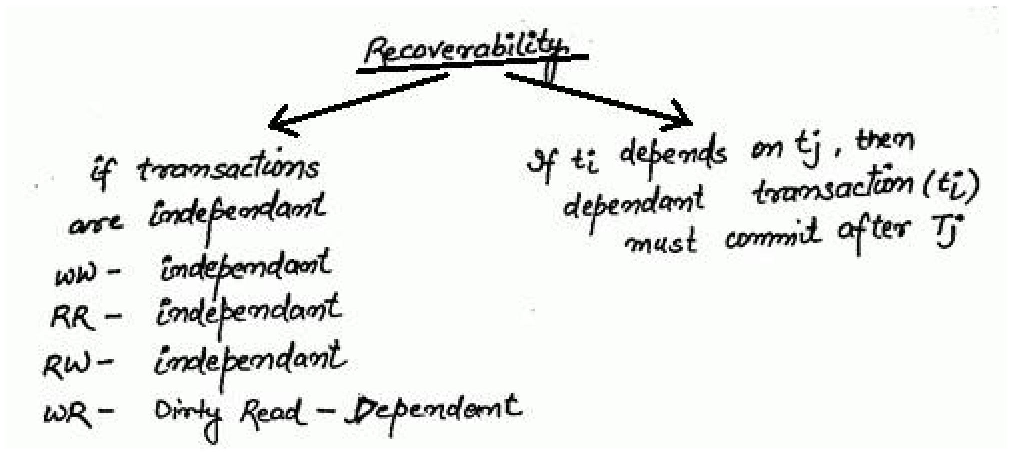
Recoverable

Case 8 :

T1	T2	T3	T4
R1(A)			
W1(A)			
	R2(X)		
	W2(X)		
		R3(X)	
		W3(X)	
			R4(X)
			W4(X)
Abort			

Recoverable

Case 5 is an irrecoverable Schedule because T2 reads a updated value before the commit operation of T1.
Case 6 is a recoverable schedule because the transaction is not reading any updated value.
Case 7 is also a recoverable schedules because commit of transaction T2 (which is reading the updated value) is to be delayed until the transaction T1 is committed or aborted.
Case 8 is a recoverable schedule because the commit of transaction T2, T3 and T4 (which are reading the updated value one by one) is to be delayed until the transaction T1 is aborted.

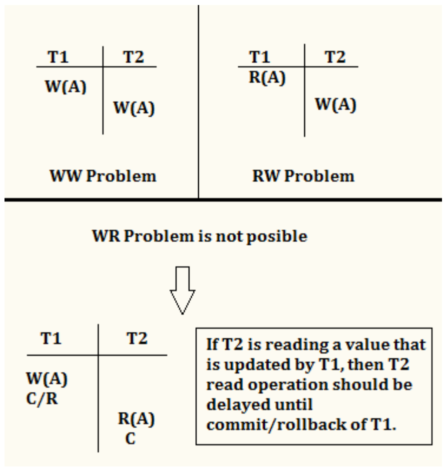

Recoverability
if transactions are independant
WW - independant
RR - independant
RW - independant
WR - Dirty Read - Dependant
If ti depends on tj, then dependant transaction (ti) must commit after Tj

Cascading Schedule or What is Cascading Rollback ?

The failure of single transaction leads to rollback of several transactions [i.e. rollback of dependent transaction]. It leads to wastage of CPU time. **For example** T2 depends on T1, T3 depends on T2, and T4 depends on T3. Hence transitively T4 on T1. If T1 fails, we have to rollback T1, then rollback T2, then T3 and finally T4. Such a rollback is known as cascading rollback (failure of transaction causes rollback of other dependent transaction). The Schedule is irrecoverable T2, T3, T4 commit before T1.

Cascadeless Schedule :

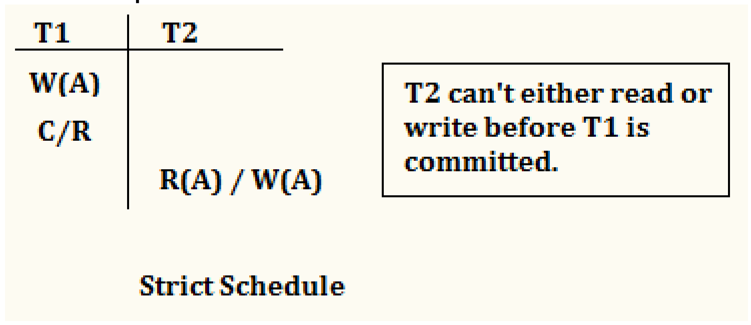
Definition 1 : A Schedule that avoids cascading rollback, is called as cascadeless schedule.
Definition 2 : ASchedule in which for every pair of transaction Ti and Tj such that read data item previously written by Ti, the commit / abort operation of Ti should appear before read operation of Tj. **Cascadeless schedule guarantees only committed read operations.** Cascadeless rollback is not free from inconsistency. It may contain
RW problem
WW problem
But the WR problem is not possible because we ensured committed read only.For Example


T1
T2
W(A)
W(A)
WW Problem
T1
T2
R(A)
W(A)
RW Problem
WR Problem is not posible
T1
T2
W(A)
C/R
R(A)
C
If T2 is reading a value that is updated by T1, then T2 read operation should be delayed until commit/rollback of T1.

Strict Schedule

A Schedule, in which a transaction neither read or write a data item X until the last transaction that has written X is committed or aborted.

For example :

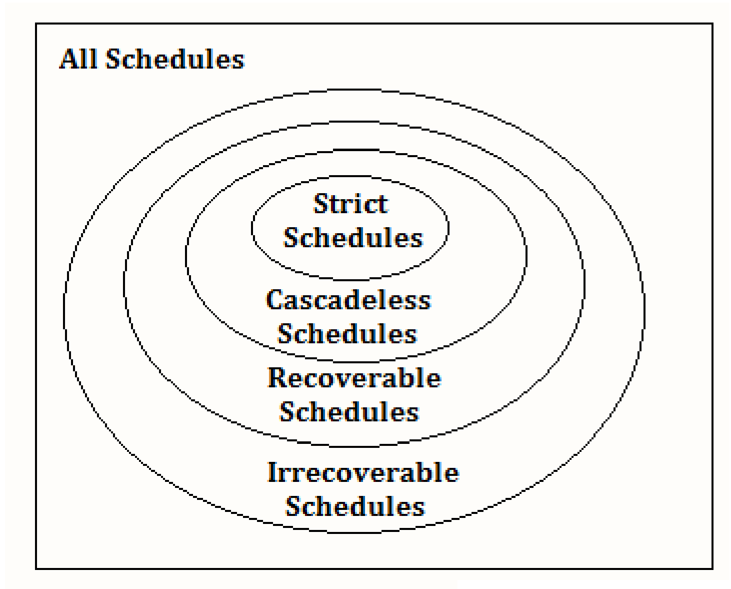


Problems in Strict Schedule :

Inconsistency is possible because of RW problem.

No inconsistency exist because of WR, WW problems.

Relation between the Schedules :



How to check for Conflict Serializable Schedule ?

Conflict Equivalence and Conflict Serializable Schedule

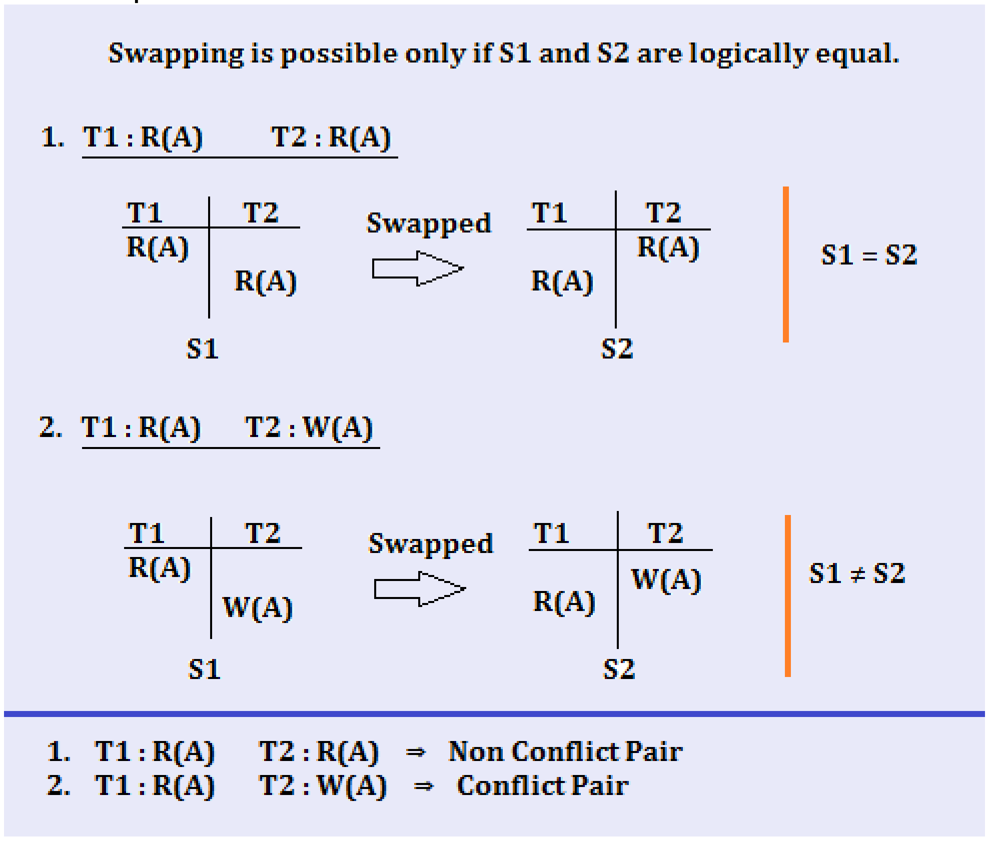
Before we discuss conflict equivalence and conflict serializable schedule, you must know about conflicts. So, what is a conflict is described in the following section.

What is a conflict?

A pair of Operations in a schedule such that if their order is interchanged then the behavior of atleast one of the transactions may change. Operations are conflict, if they satisfy all three of the following conditions :

- They belong to different transactions
- They access the same data item
- At least one of the operation is a write operation.

For example :

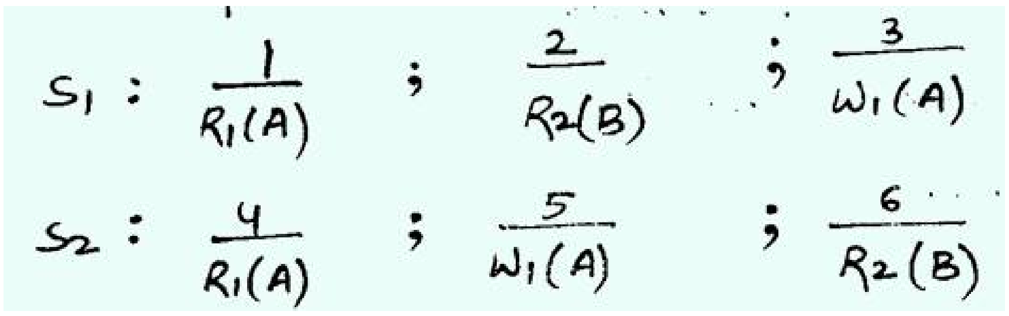


1. R1(A);W2(A) :	It is a conflict pair, Because T1 is reading initial value of A and T2 writing a different value for A and if we interchange their order then T1 will read the value of A written by T2 , thus behavior will change.
2. W1(A); R2(A) 3. W1(A); W2(A)	Conflict Pairs
4. R1(A); R2(A)	Not a Conflict Pair
5. R1(A); W2(B) 6. W1(A); R2(B)	Not a Conflict Pairs because they are acting on two different values, A and B.
7. R1(A); W1(A)	We can't interchange order within same transaction. Therefore, No conflict

Conflict equivalence

Schedules are conflict equivalent if they can be transformed one into other by a sequence of non conflicting interchanges adjacent actions.

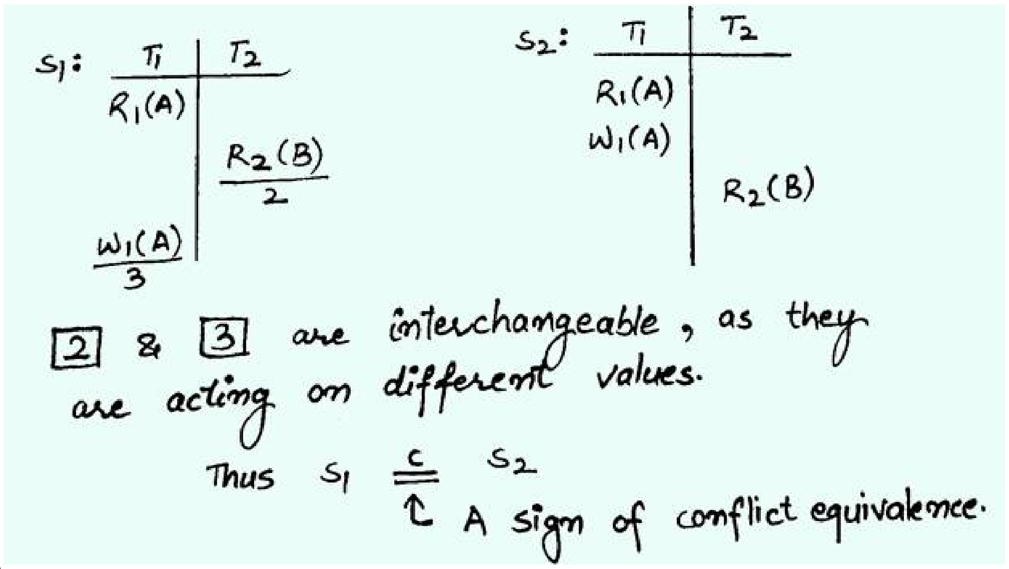
For example :
Question 1 : Check whether the schedules S1 and S2 are conflict equivalent or not ?



$S_1 : \underset{R_1(A)}{\overset{1}{\_\_}} ; \underset{R_2(B)}{\overset{2}{\_\_}} ; \underset{W_1(A)}{\overset{3}{\_\_}}$

$S_2 : \underset{R_1(A)}{\overset{4}{\_\_}} ; \underset{W_1(A)}{\overset{5}{\_\_}} ; \underset{R_2(B)}{\overset{6}{\_\_}}$

Solution :



S1:

T1	T2
R1(A)	
	R2(B) 2
W1(A) 3	

S2:

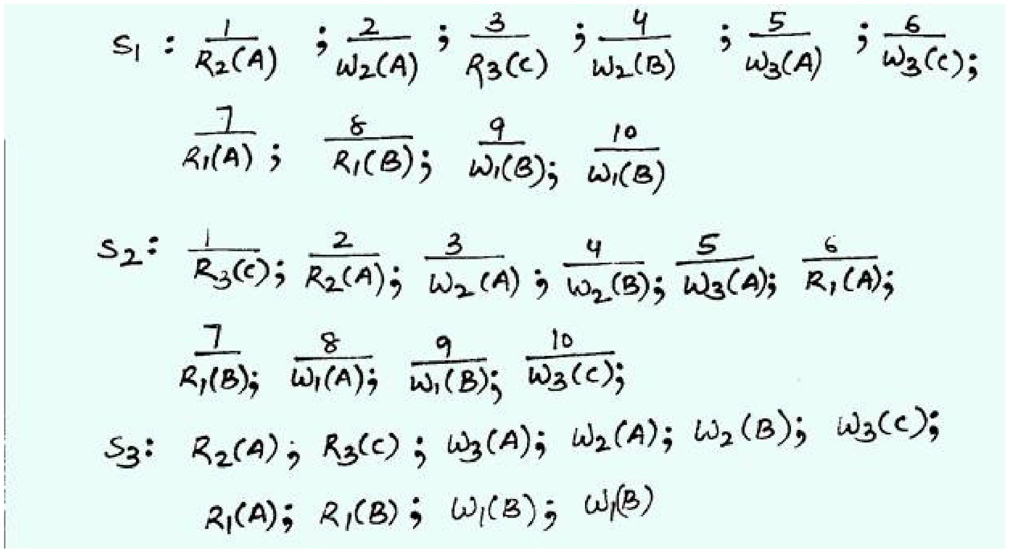
T1	T2
R1(A)	
W1(A)	
	R2(B)

$\boxed{2}$ & $\boxed{3}$ are interchangeable , as they are acting on different values.

Thus $S_1 \overset{c}{=} S_2$

↑ A sign of conflict equivalence.

Question 2 : Check whether the schedules S1, S2 and S3 are conflict equivalent or not ?



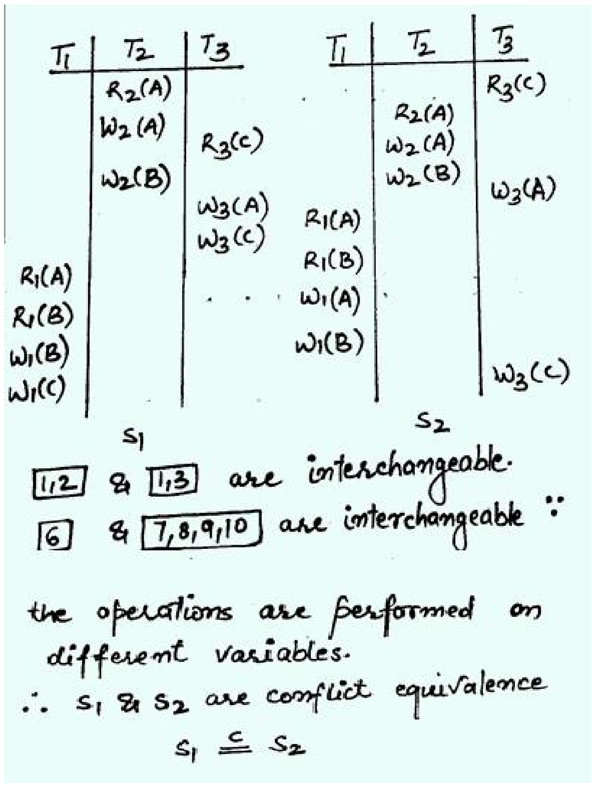
S_1 : $\frac{1}{R_2(A)}$; $\frac{2}{W_2(A)}$; $\frac{3}{R_3(C)}$; $\frac{4}{W_2(B)}$; $\frac{5}{W_3(A)}$; $\frac{6}{W_3(C)}$;
$\frac{7}{R_1(A)}$; $\frac{8}{R_1(B)}$; $\frac{9}{W_1(B)}$; $\frac{10}{W_1(B)}$

S_2 : $\frac{1}{R_3(C)}$; $\frac{2}{R_2(A)}$; $\frac{3}{W_2(A)}$; $\frac{4}{W_2(B)}$; $\frac{5}{W_3(A)}$; $\frac{6}{R_1(A)}$;
$\frac{7}{R_1(B)}$; $\frac{8}{W_1(A)}$; $\frac{9}{W_1(B)}$; $\frac{10}{W_3(C)}$;

S_3 : $R_2(A)$; $R_3(C)$; $W_3(A)$; $W_2(A)$; $W_2(B)$; $W_3(C)$;
$R_1(A)$; $R_1(B)$; $W_1(B)$; $W_1(B)$

Solution :

Check for S1 and S2 are conflict equivalent or not ?



T_1	T_2	T_3
	$R_2(A)$	
	$W_2(A)$	
		$R_3(C)$
	$W_2(B)$	
		$W_3(A)$
		$W_3(C)$
$R_1(A)$		
$R_1(B)$		
$W_1(B)$		
$W_1(C)$		

S_1

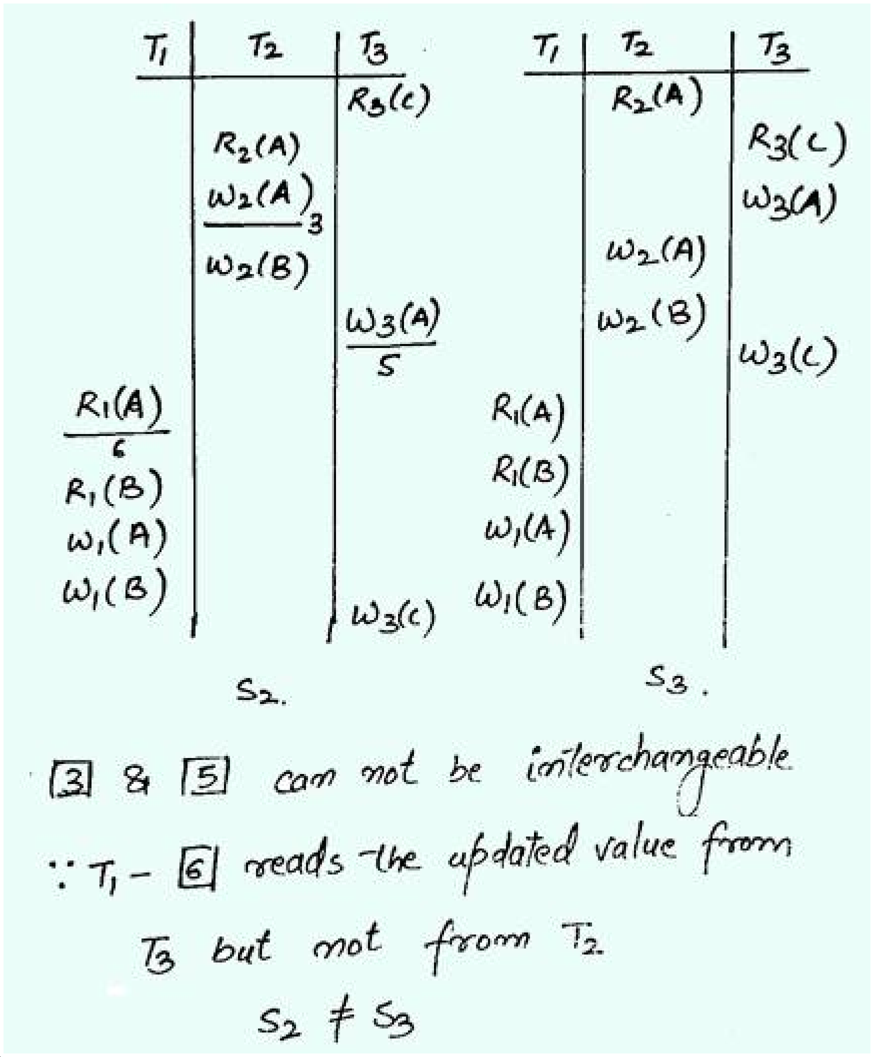
T_1	T_2	T_3
		$R_3(C)$
	$R_2(A)$	
	$W_2(A)$	
	$W_2(B)$	
		$W_3(A)$
$R_1(A)$		
$R_1(B)$		
$W_1(A)$		
$W_1(B)$		
		$W_3(C)$

S_2

$\boxed{1,2}$ & $\boxed{1,3}$ are interchangeable.
$\boxed{6}$ & $\boxed{7,8,9,10}$ are interchangeable ∵

the operations are performed on different variables.
∴ S_1 & S_2 are conflict equivalence
$S_1 \overset{c}{=} S_2$

Check for S2 and S3 are conflict equivalent or not?



T_1	T_2	T_3
		$R_3(C)$
	$R_2(A)$	
	$\underline{W_2(A)}$ 3	
	$W_2(B)$	
		$\underline{W_3(A)}$ 5
$\underline{R_1(A)}$ 6		
$R_1(B)$		
$W_1(A)$		
$W_1(B)$		
		$W_3(C)$

S_2.

T_1	T_2	T_3
	$R_2(A)$	
		$R_3(C)$
		$W_3(A)$
	$W_2(A)$	
	$W_2(B)$	
		$W_3(C)$
$R_1(A)$		
$R_1(B)$		
$W_1(A)$		
$W_1(B)$		

S_3.

$\boxed{3}$ & $\boxed{5}$ can not be interchangeable

$\because T_1$ - $\boxed{6}$ reads the updated value from T_3 but not from T_2.

$S_2 \neq S_3$

Conflict Serializable Schedule

A Schedule is conflict serializable if it is conflict equivalent to any of serial schedule.

Testing for conflict serializability

Method 1 :

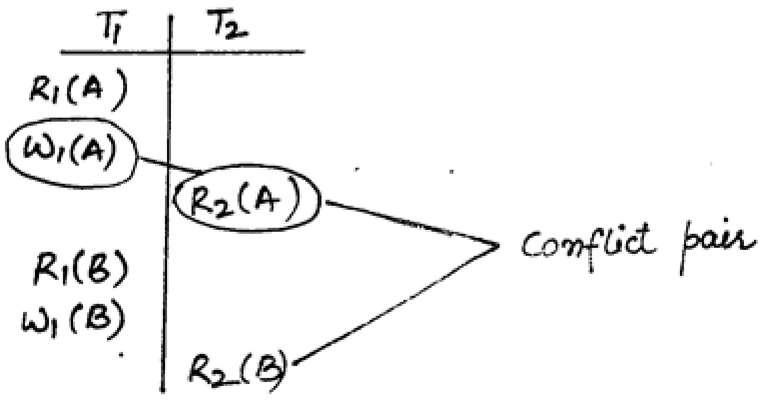
- First write the given schedule in a linear way.
- Find the conflict pairs (RW, WR, WW) on same variable by different transactions.
- Whenever conflict pairs are find, write the dependency relation like Ti → Tj, if conflict pair is from Ti to Tj. For example, (W1(A), R2(A)) ⇒ T1 → T2
- Check to see if there is a cycle formed,
    - If yes= not conflict serializable
    - No= we get a sequence and hence are conflict serializable.

Example :

Question: Check whether the schedule is conflict serializable or not?

S: R1(A); W1(A); R2(A); R1(B); W1(B); R2(B)

Solution :



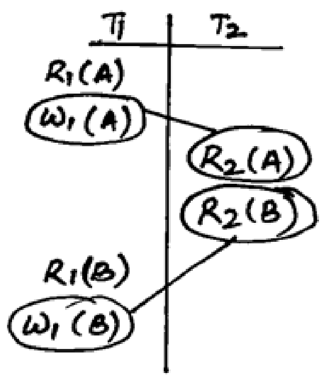
Conflict Pairs	Dependency Relation
W1(A); R2(A)	T1 → T2
W1(B); R2(B);	T1 → T2

No Cycle formed. Therefore, Conflict Serializable Schedule.

Question: Check whether the schedule is conflict serializable or not?

S: R1(A); W1(A); R2(A); R2(B); R1(B); W1(B)

Solution :



CONFLICT PAIRS:-

$W_1(A); R_2(A)$

$R_2(B); W_1(B)$

DEPENDENCY RELATION:-

$T_1 \rightarrow T_2$

$T_2 \rightarrow T_1$

cycle dependency

Hence cycle formed ∴ Not a Conflict serializable schedule.

Method 2:

To test the conflict serializability, we can draw a Graph G = (V,E) where V = Vertices = number of transactions E = Edges = for conflicting pair

Steps :

1. Create node for each transaction.
2. Find the conflict pairs (RW, WR, WW) on same variable by different transactions.
3. Draw edge from the schedule for each conflict pair such that for example, W2(B), R1(A) is conflict pair, draw edge from T2 to T1 i.e. T2 must be executed before T1.
4. Testing conditions for conflict serializability of schedule
    - If precedence graph is cyclic non conflict serializable schedule
    - If precedence graph is a acyclic conflict serializable schedule

Example :

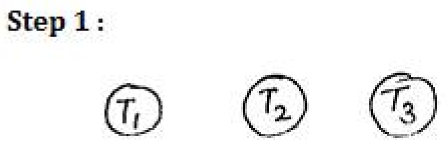
Question : Check whether the schedule is conflict serializable or not?

S: R1(A); R2(A); R1(B); R2(B); R3(B); W1(A); W2(B)

Solution :

$$S: R_1(A);\ \underset{4}{\underline{R_2(A)}};\ \underset{1}{\underline{R_1(B)}};\ R_2(B);\ \underset{2}{\underline{R_3(B)}};\ \underset{b}{\underline{W_1(A)}};\ \underset{a}{\underline{W_2(B)}};$$

Step 1 : Create a node for each transaction.



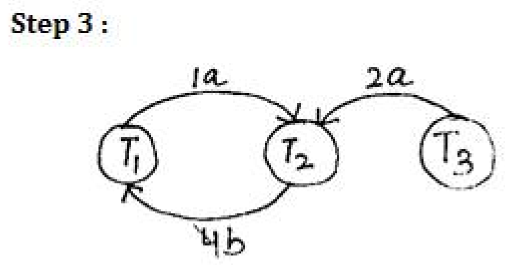
Step 2 : Find the conflict pairs (RW, WR, WW) on same variable by different transactions.

R3(B); W2(B)

R1(B); W2(B)

R2(A); W1(A)

Step 3 : Draw an edge for each conflict pair.



As Cycle formed, Therefore Schedule is not a conflict serializable schedule.

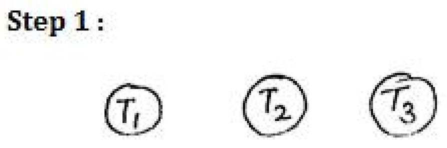
Question : Check whether the schedule is conflict serializable or not?

S: R1(A); R2(A); R3(B); W1(A); R2(C); R2(B); W2(B);

Solution :

$$S: R_1(A);\ \underset{2}{\underline{R_2(A)}};\ \underset{1}{\underline{R_3(B)}};\ \underset{b}{\underline{W_1(A)}};\ R_2(C);\ R_2(B);\ \underset{a}{\underline{W_2(B)}};$$

Step 1 : Create a node for each transaction.



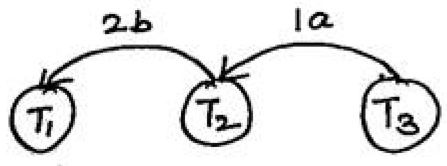
Step 2 : Find the conflict pairs (RW, WR, WW) on same variable by different transactions.

R3(B); W2(B)

R2(A); W1(A)

Step 3 : Draw an edge for each conflict pair.

Step 3 :



No cycle formed. Hence schedule is conflict serializable schedule and the equivalent serial schedule is T3 → T2 → T1.

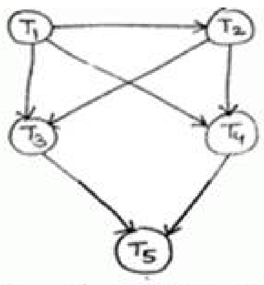
Find Number of Equivalent Serial Schedule Using Topological Order
G = (V,E)

where V = Vertices : Transactions of the schedule
E = Edges : Conflict Pair Precedence Order

How Topological Order Finds out the Number of Conflict Equivalent Serial Schedule ??

1. Visit vertex(V) with indegree (0) and delete V and all the edges going to other vertex from V in the G.
2. Repeat (1) until G becomes Empty.
3. Number of Conflict Equivalent serial schedule is number of topological orders of acyclic precedence graph.

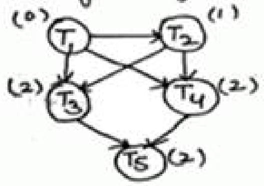
Example :



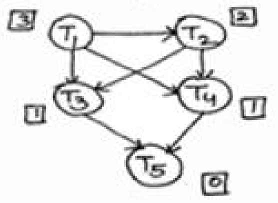
Find indegree of : Check how many edges are coming
vertex V to vertex V.

Find outdegree of : Check how many edges are going
Vertex V out from letter V.

~~Ing.~~ Indegree

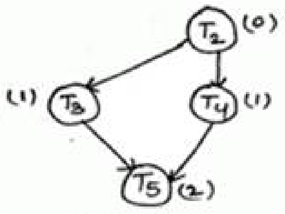


Out degree

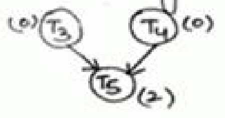


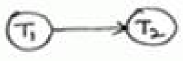
( ): Indegree of Vertex
□ : Outdegree of Vertex

Step 1 : Deleting vertex of indegree (0) & all the edges going out to other vertices.
⇒ Deleting T_1 from G & updating indegree of remaining vertices.



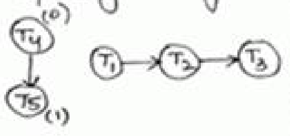
Step 2 : Deleting T_2 from G & updating indegree of remaining vertices.



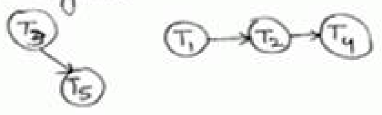


Step 3: Now there are two possible topological orders (∵ there are two vertices having indegree (0))

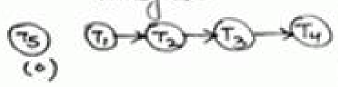
(i) ~~@~~ Deleting T_3 & updating indegrees.



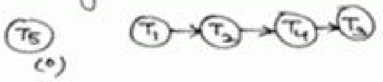
(ii) Deleting T_4 & updating indegrees.



Step4 : Deleting & updating indegrees



Deleting T_3 & updating indegrees.



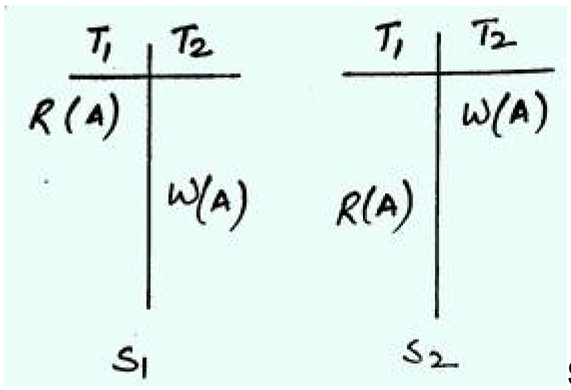
How to check for View Serializable Schedule ?

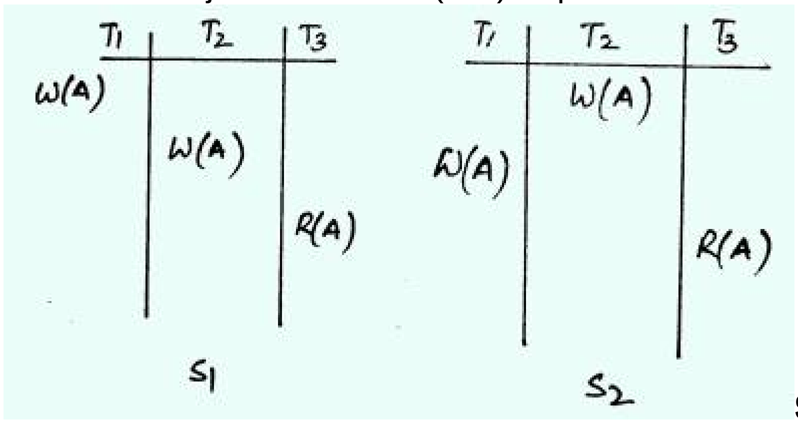
Before we discuss about view serializable schedule, we must know about view equivalent schedule.

View equivalent schedule and View Serializable Schedule

View Equivalent Schedule :

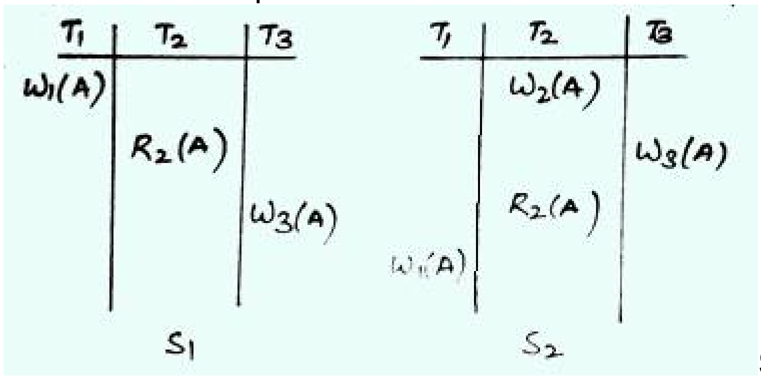
Consider two schedules S1 and S2, they are said to be view equivalent if following conditions are true :

- Initial read must be same. 

S1 : T1 reads A from Database. S2 : T1 reads A from T2. ∴ S1 ≠ S2.
- There are two transactions say Ti and Tj, The schedule S1 and S2 are view equivalent if in schedule S1, Ti reads A that has been updated by Tj, and in schedule S2, Ti must read A from Tj. i.e. write-read(WR) sequence must be same between S1 and S2. 

S1 : T3 reads value of A from T2. S2 : T3 reads value of A from T1. ∴ S1 ≠ S2. i.e. write-read sequence is not same between S1 and S2.

- Final write operations should be same between S1 and S2.



S1 : A is finally updated by T3. S2 : A is finally updated by T1. ∴ S1 ≠ S2.

View serializable schedule

A Schedule is view serializable if it is view equivalent to any serial schedule. The following two examples will illustrate how to find view equivalent schedule. **Examples :**
Question 1 : Check whether the schedule is view serializable or not?
S : R2(B); R2(A); R1(A); R3(A); W1(B); W2(B); W3(B);
Solution : With 3 transactions, total number of schedules possible
= 3! = 6
<T1 T2 T3>
<T1 T3 T2>
<T2 T3 T1>
<T2 T1 T3>
<T3 T1 T2>
<T2 T2 T1>

Step 1 : Final Updation on data items :
A : -
B : T1 T2 T3
Since the final updation on B is made by T3, so the transaction T3 must execute before transactions T1 and T2.
Therefore, (T1,T2) → T3
Now, Removing those schedules in which T3 is not executing at last.

Remaining Schedules :
<T1 T2 T3>
<T2 T1 T3>

Step 2 : Initial Read + Which transaction updates after read?

A : T2 T1 T3

B : T2 T1

The transaction T2 reads B initially which is updated by T1. So T2 must execute before T1.

Hence, T2 → T1

Now, Removing those schedules in which T2 is executing before T1.

Remaining Schedules :

<T2 T1 T3>

Step 3 : Write Read Sequence (WR) :

No need to check here.

Hence, view equivalent serial schedule is :

T2 → T1 → T3

Question 2 : Check whether the schedule is Conflict serializable and view serializable or not?

S : R1(A); R2(A); R3(A); R4(A); W1(B); W2(B); W3(B); W4(B)

Solution :

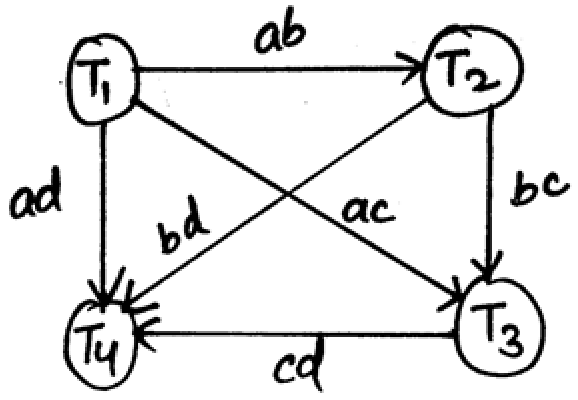
Check for Conflict Equivalent Schedule :

Step 1 : Create a node for each transaction.

Step 2 : Find the conflict pairs (RW, WR, WW) on same variable by different transactions.

W1(B); W2(B)
W1(B); W3(B)
W1(B); W4(B)
W2(B); W3(B)
W2(B); W4(B)
W3(B); W4(B)

Step 3 : Draw an edge for each conflict pair.



-

No Cycle Formed. Hence it is conflict serializable schedule and the conflict serial schedule is

T1 → T2 → T3 → T4

Note : Only 1 conflict equivalent schedule is possible.

<u>Check for View Equivalent Schedule</u> :

Step 1 : Final Updation on data items :

A : -

B : T1 T2 T3 <u>T4</u>

Since the final updation on B is made by T4, so the transaction T4 must execute after all the transactions.

i.e. (T1,T2,T3) → T4

Step 2 : <u>Initial Read</u> + <u>Which transaction updates after read</u>?

A : <u>T1</u> T2 T3 T4

B : -

No dependency can be derived from step 2.

Step 3 : Write Read Sequence (WR) :

No WR sequence is in the schedule S.

Hence, the remaining dependency is

(T1 T2 T3) → T4

The ways we can arrange (T1 T2 T3) = 3! = 6 ways.

Hence, Total View Equivalent Serial Schedule possible = 6

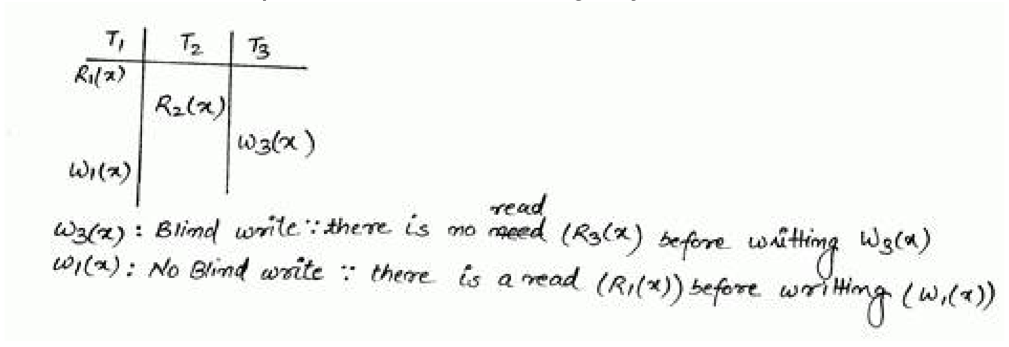
Method 2 to find view Serializable : Polygraph

- Place an edge for each read operation indicating its source.
- Place the edges for writers of data items to indicate the possible interference.
- Testing Conditions :
    - If a cycle is formed, then it is not view serializable.
    - If no cycle is formed, then it is view serializable.

A Shortcut for Finding View Serializable : Funda of Blind Write :

What is a Blind Write ??

If there is no read operation before writing any value then it is Blind Write. For Example



T_1	T_2	T_3
$R_1(x)$		
	$R_2(x)$	
		$W_3(x)$
$W_1(x)$		

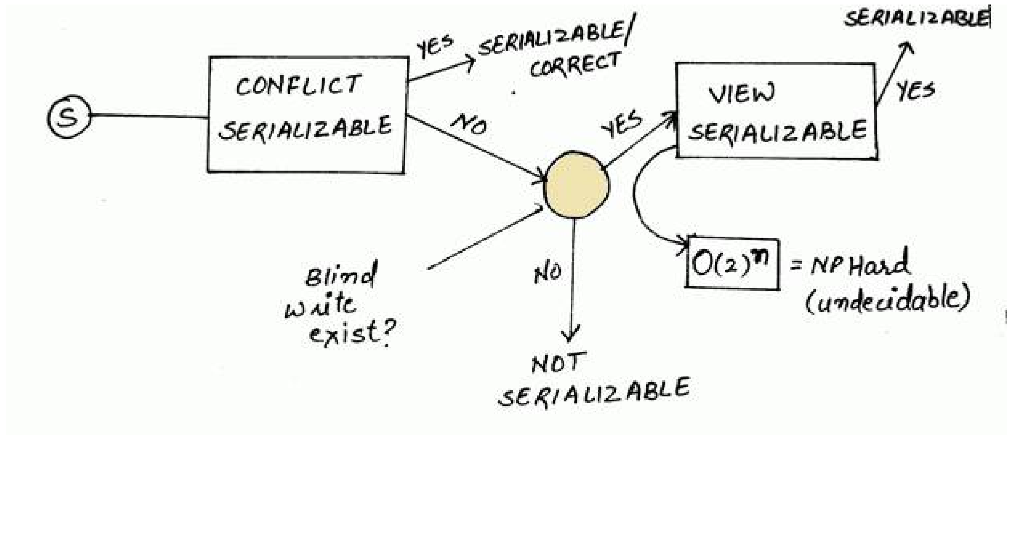
$W_3(x)$: Blind write ∵ there is no ~~need~~ read $(R_3(x))$ before writting $W_3(x)$

$W_1(x)$: No Blind write ∵ there is a read $(R_1(x))$ before writting $(W_1(x))$

:

How to Check For View Serializable Using Blind Write ??

Any schedule that is not Conflict Serializable but View Serializable should contain atleast one Blind Write Operation i.e. If a Blind Write Operation in a schedule does not exist, then the Schedule is **NOT SERIALIZABLE**.



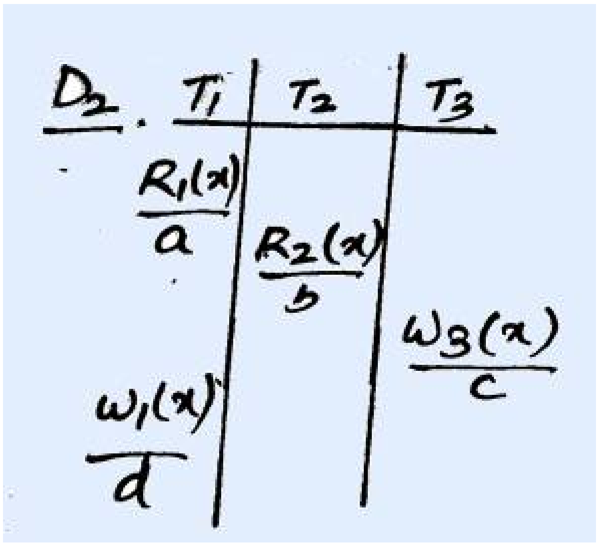
Necessary and Sufficient Condition :

A Schedule is serializable, If it is conflict or view equivalent to any serial schedule.

- Sufficient Condition : Conflict Serializability is sufficient condition for a serializable schedule.
- Necessary & Sufficient Condition : View Serializability is both Necessary and Sufficient Condition for Serializable Schedule. →

Example :

Question 1 : Check whether the schedule is view serializable or not ?

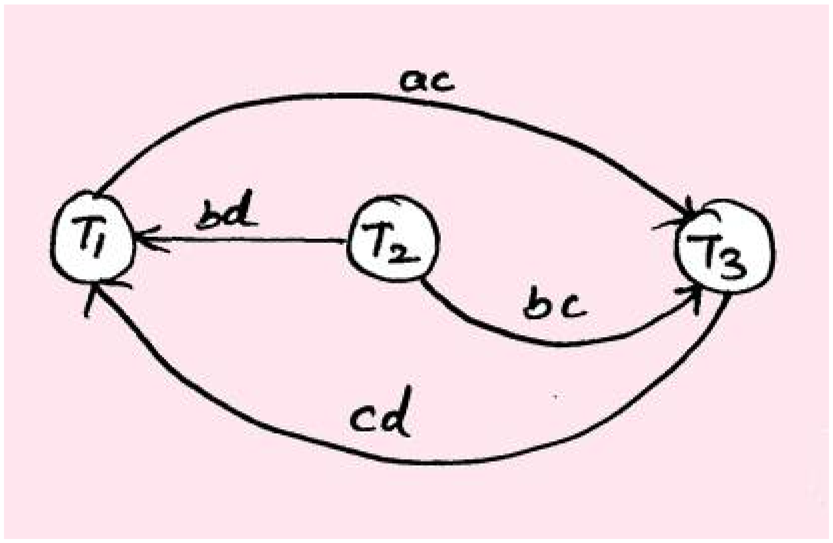


D2.	T1	T2	T3
	$R_1(x)$ a		
		$R_2(x)$ b	
			$W_3(x)$ c
	$W_1(x)$ d		

Solution :

Step 1 : Check for Conflict Serializable :

Conflicts :	Dependency :
R1(X); W3(X)	T1 → T3 [ac]
R2(X); W3(X)	T2 → T3 [bc]
R2(X); W1(X)	T2 → T1 [bd]
W3(X); W1(X)	T3 → T1 [cd]



Step 2 : Check For <u>Blind Write</u> :

Since W3(x) is a blind write, therefore we will check for view serializability.

Step 3 : Check For <u>View Serializability</u> :

(i) Final Updation on Data Items:

<u>Dependency</u> :

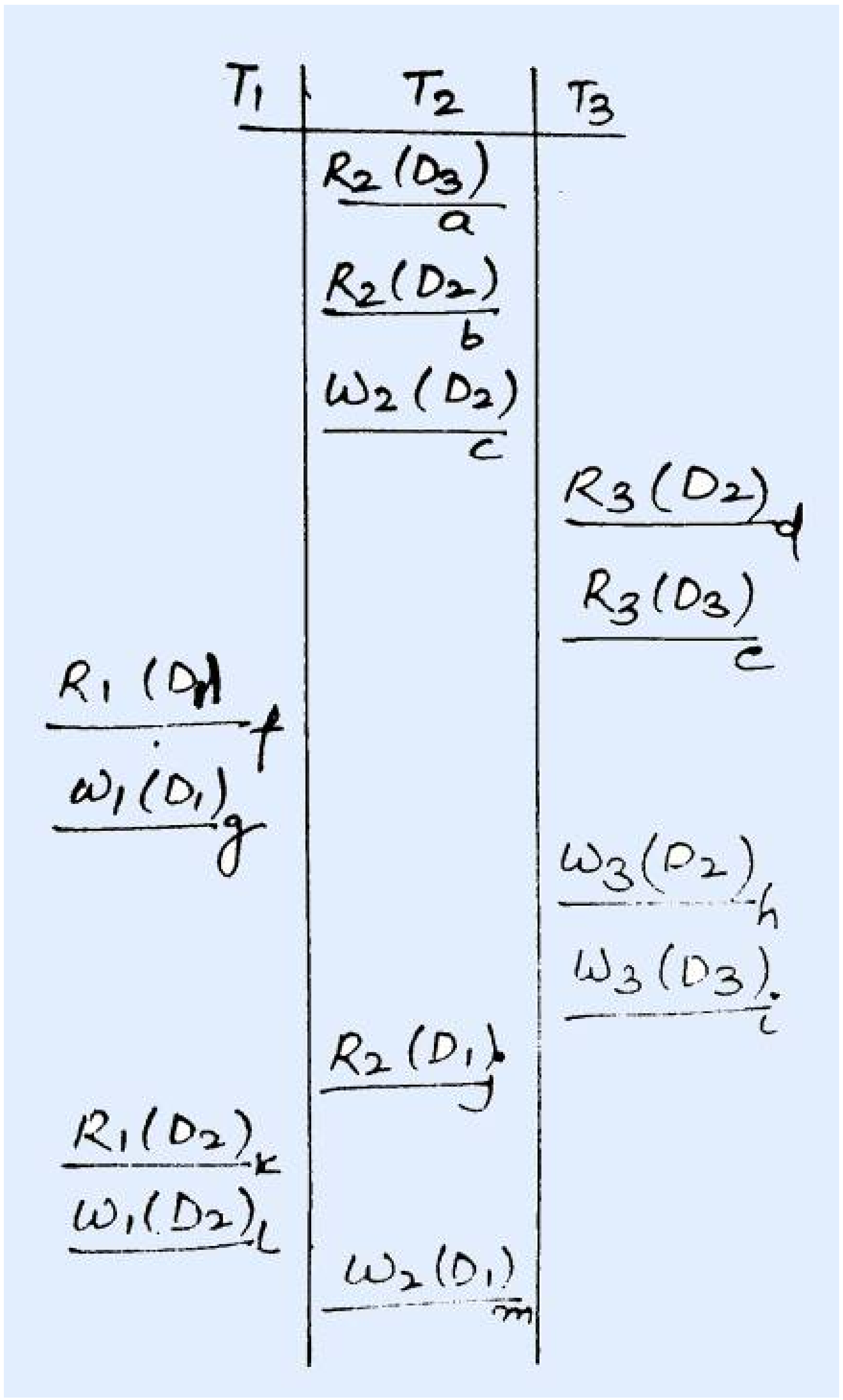
x: T3 T1 $T3 \rightarrow T1$ .....(p)

(ii) Initial Read :

<u>Initial Read</u> :	<u>Updation</u> :	<u>Dependency</u> :
x : T1 T2	T3 T1	$T1 \rightarrow T3$; $T2 \rightarrow T3$
		$\Rightarrow (T1\ T2) \rightarrow T3$ .....(q)

From Dependency (p), T3 must execute before T1 and From Dependency (q) T1 must execute before T3. Hence cycle formed. Therefore, the schedule is not View serializable.

Question 2 : Check whether the schedule is view serializable or not ?

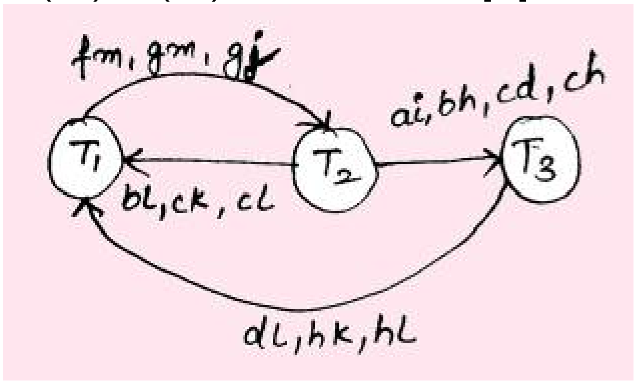


T_1	T_2	T_3
	$R_2(D_3)$ a	
	$R_2(D_2)$ b	
	$W_2(D_2)$ c	
		$R_3(D_2)$ d
		$R_3(D_3)$ e
$R_1(D_1)$ f		
$W_1(D_1)$ g		
		$W_3(D_2)$ h
		$W_3(D_3)$ i
	$R_2(D_1)$ j	
$R_1(D_2)$ k		
$W_1(D_2)$ l		
	$W_2(D_1)$ m	

Solution :

Step 1 : Check for Conflict Serializable :

Conflicts :	Dependency :
R2(D3) W3(D3)	T2 → T3 [ai]
R2(D2) W3(D2)	T2 → T3 [bh]
R2(D2) W1(D2)	T2 → T1 [bl]
W2(D2) R3(D2)	T2 → T3 [cd]
W2(D2) W3(D2)	T2 → T3 [ch]
W2(D2) R1(D2)	T2 → T1 [ck]
W2(D2) W1(D2)	T2 → T1 [cl]
R3(D2) W1(D2)	T3 → T1 [dl]
R1(D1) W2(D1)	T1 → T2 [fm]
W1(D1) R2(D1)	T1 → T2 [gj]
W1(D1) W2(D1)	T1 → T2 [gm]
W3(D2) R1(D2)	T3 → T1 [hk]
W3(D2) W1(D2)	T3 → T1 [hl]


$$\begin{tikzcd}
T_1 \arrow[r, bend left, "{fm, gm, gj}"] & T_2 \arrow[l, "{bl, ck, cl}"] \arrow[r, "{ai, bh, cd, ch}"] & T_3 \arrow[ll, bend left, "{dl, hk, hl}"]
\end{tikzcd}$$

Step 2 : Check for Blind Write :

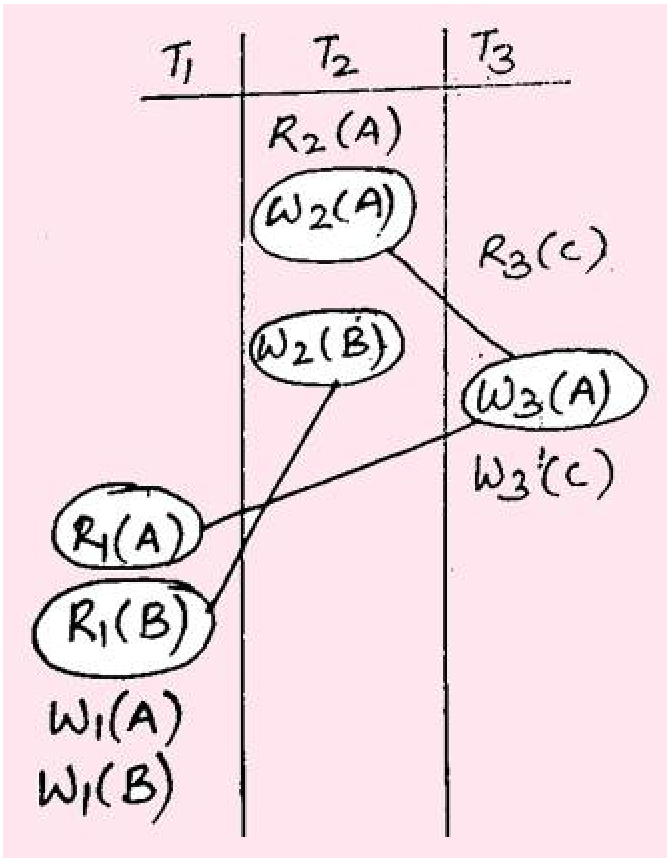
Since there is a read operation for every write operation of variable. Thus no blind write exists.
Therefore not View Serializable also.

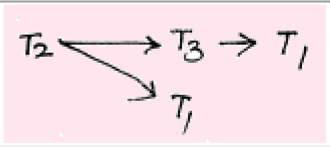
Questions on Conflict Serializable

Question 1 : Check whether the schedules is conflict serializable or not ?

S : R2(A); W2(A); R3(C); W2(B); W3(A); W3(C); R1(A);R1(B); W1(A); W1(B)

Solution :



Conflict Pairs	Dependency Relation
W2(A) W3(A)	T2 → T3
W3(A) R1(A)	T3 → T1
W2(B) R1(B)	T2 → T1
	

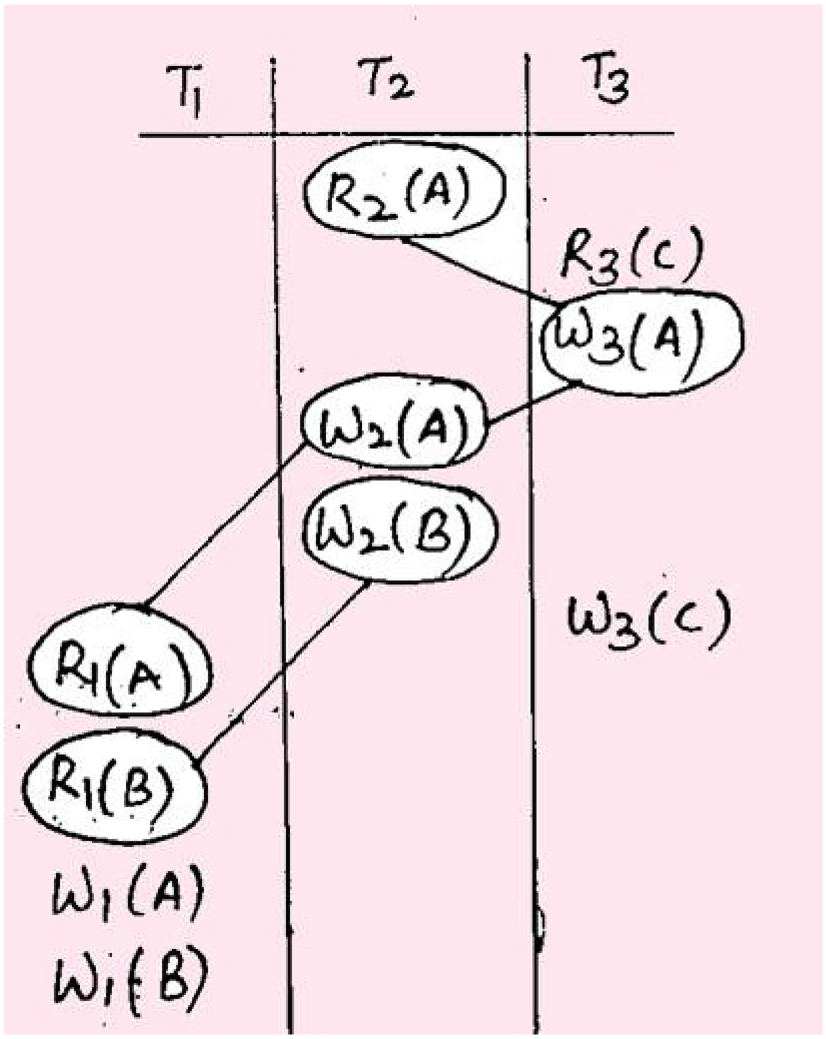
$$\begin{tikzcd} T_2 \arrow[r] \arrow[dr] & T_3 \arrow[r] & T_1 \\ & T_1 & \end{tikzcd}$$

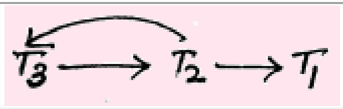
⇒ No cycle formed. Therefore, conflict serializable schedule and the equivalent serial schedule is
T2 → T3 → T1.

Question 2 : Check whether the schedule are conflict serializable or not ?

S : R2(A); R3(C); W3(A); W2(A); W2(B); W3(C); R1(A); R1(B); W1(A); W1(B)

Solution :



Conflict Pairs	Dependency Relation
R1(A) W3(A)	T2 → T3
W3(A) W2(A)	T3 → T2
W2(A) R1(A)	T2 → T1
W2(B) R1(B)	T2 → T1
	

⇒ No cycle formed. Therefore, conflict serializable schedule and the equivalent serial schedule is
T2 → T3 → T1.

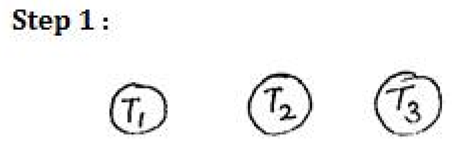
Question 3 : Check whether the schedule is conflict serializable or not?

S: R1(A); R2(A); R3(B); W1(A); R2(C); R2(B); W2(B); W1(C)

Solution :

$$S:\ R_1(A)\quad \underset{3}{\underline{R_2(A)}}\quad \underset{2}{\underline{R_3(B)}}\quad \underset{c}{\underline{W_1(A)}}\quad \underset{1}{\underline{R_2(C)}}\quad R_2(B)\quad \underset{b}{\underline{W_2(B)}}\quad \underset{a}{\underline{W_1(C)}}$$

Step 1 : Create a node for each transaction.



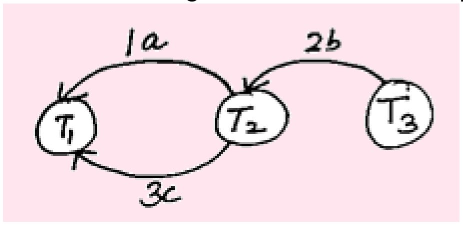
Step 2 : Find the conflict pairs (RW, WR, WW) on same variable by different transactions.

R2(C); W1(C)

R3(B); W2(B)

R2(A); W1(A)

Step 3 : Draw an edge for each conflict pair.



As Cycle formed, Therefore Schedule is not a conflict serializable schedule.

Question 4 : Check whether the schedule is conflict serializable or not?

S: W3(A); R1(A); W1(B); R2(B); W2(C); R3(C)

Solution :

$$S:\ \underset{3}{\underline{W_3(A)}}\quad \underset{1}{\underline{R_1(A)}}\quad \underset{2}{\underline{W_1(B)}}\quad \underset{b}{\underline{R_2(B)}}\quad \underset{1}{\underline{W_2(C)}}\quad \underset{a}{\underline{R_3(C)}}$$

Step 1 : Create a node for each transaction.
(performed in question 3)

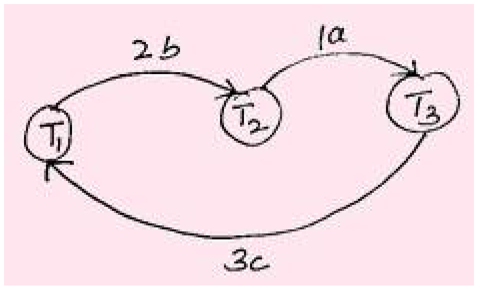
Step 2 : Find the conflict pairs (RW, WR, WW) on same variable by different transactions.

W2(C); R3(C)

W1(B); R2(B)

W3(A); R1(A)

Step 3 : Draw an edge for each conflict pair.

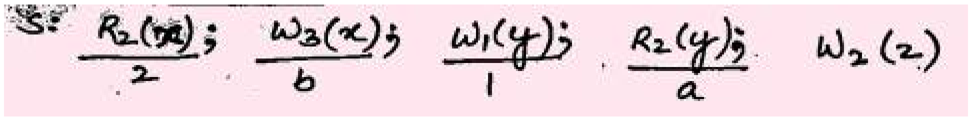


As Cycle formed, Therefore Schedule is not a conflict serializable schedule.

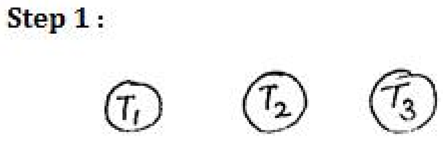
Question 5 : Check whether the schedule is conflict serializable or not?

S: R2(x); W3(x); W1(y); R2(y); W2(z)

Solution :


S: R2(x); W3(x); W1(y); R2(y); W2(z)
2
b
1
a

Step 1 : Create a node for each transaction.

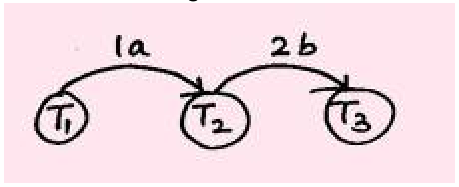


Step 2 : Find the conflict pairs (RW, WR, WW) on same variable by different transactions.

R2(x); W3(x)

W1(y); R2(y)

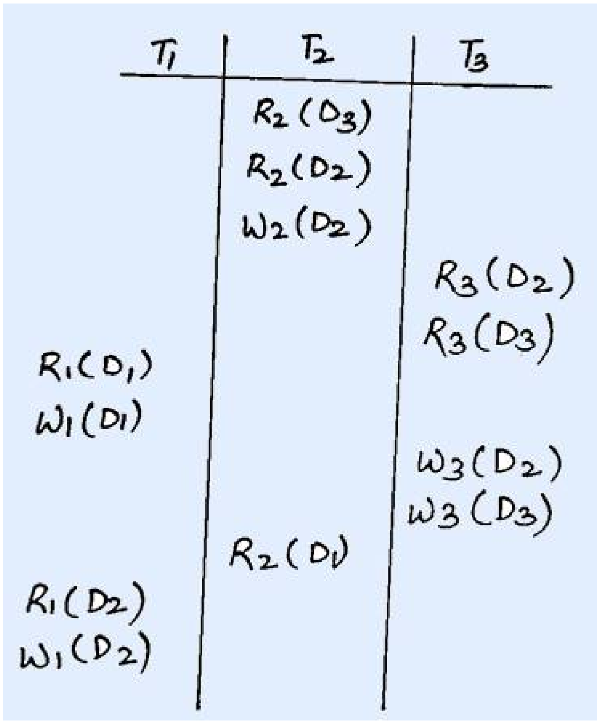
Step 3 : Draw an edge for each conflict pair.



As No Cycle formed, Therefore Schedule is conflict serializable and equivalent serial schedule is T1 → T2 → T3.

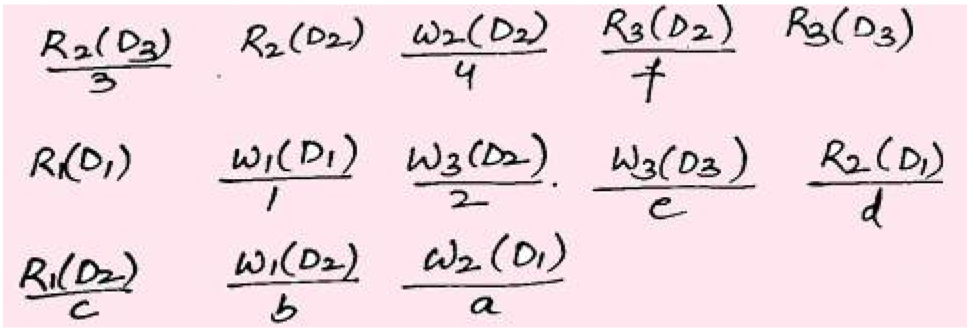
Question 6 : Consider three data items D1,D2, and D3, and the following execution schedule of transactions T1, T2, and T3. In the diagram, R(D) and W(D) denote the actions reading and writing the data item D respectively.

S: R2(D3); R2(D2); W2(D2); R3(D2); R3(D3); R1(D1);
W1(D1); W3(D2); W3(D3); R2(D1); R1(D2); W1(D2);
W2(D1), **W2(D1)**



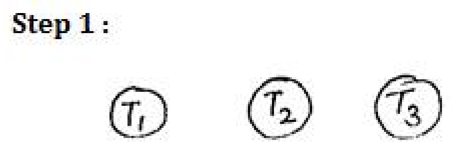
T_1	T_2	T_3
	$R_2(D_3)$	
	$R_2(D_2)$	
	$W_2(D_2)$	
		$R_3(D_2)$
		$R_3(D_3)$
$R_1(D_1)$		
$W_1(D_1)$		
		$W_3(D_2)$
		$W_3(D_3)$
	$R_2(D_1)$	
$R_1(D_2)$		
$W_1(D_2)$		

Solution :



$\underset{3}{\underline{R_2(D_3)}}$	$R_2(D_2)$	$\underset{4}{\underline{W_2(D_2)}}$	$\underset{f}{\underline{R_3(D_2)}}$	$R_3(D_3)$
$R_1(D_1)$	$\underset{1}{\underline{W_1(D_1)}}$	$\underset{2}{\underline{W_3(D_2)}}$	$\underset{e}{\underline{W_3(D_3)}}$	$\underset{d}{\underline{R_2(D_1)}}$
$\underset{c}{\underline{R_1(D_2)}}$	$\underset{b}{\underline{W_1(D_2)}}$	$\underset{a}{\underline{W_2(D_1)}}$		

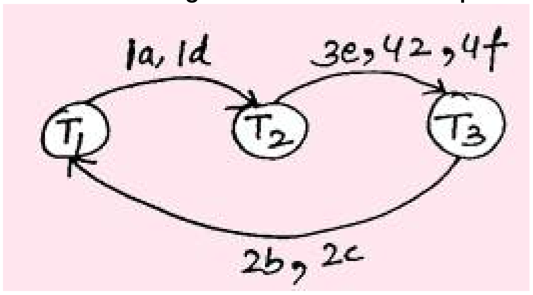
Step 1 : Create a node for each transaction.



Step 2 : Find the conflict pairs (RW, WR, WW) on same variable by different transactions.

W1(D1); W2(D1) [1a]
W1(D1); R2(D1) [1d]
R2(D3); W3(D3) [3e]
W2(D2); W3(D2) [42]
W2(D2); R3(D2) [4f]
W3(D2); W1(D2) [2b]
W3(D2); R1(D2) [2c]

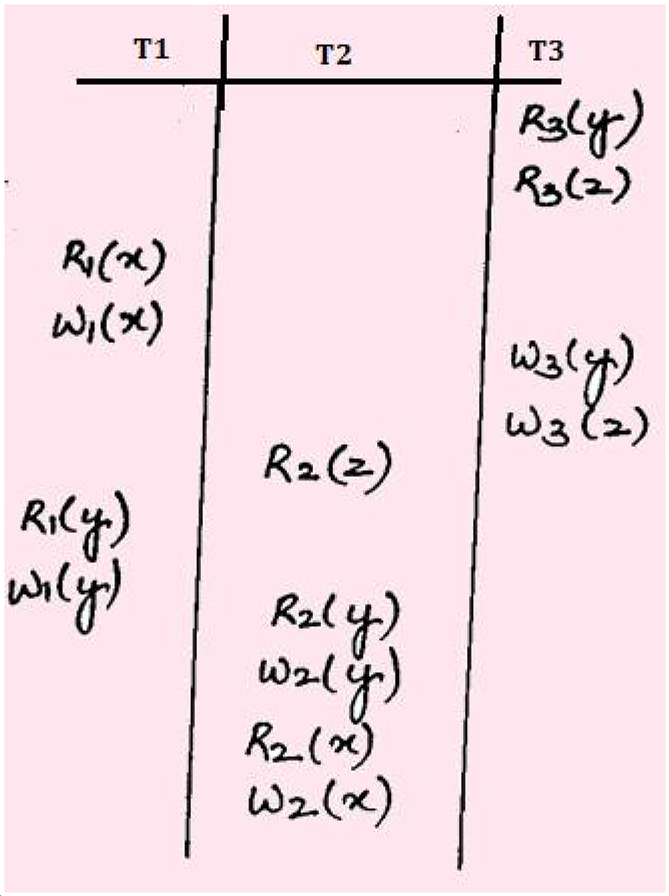
Step 3 : Draw an edge for each conflict pair.



As Cycle formed, Therefore Schedule is not a conflict serializable schedule.

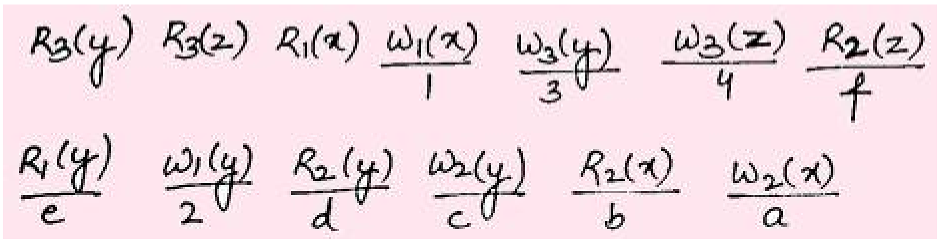
Question 7 : Check whether the schedule is conflict serializable or not?

S: R3(y); R3(z); R1(x); W1(x); W3(y); W3(z); R2(z);
R1(y); W1(y); R2(y); W2(y); R2(x); W2(x)



T1	T2	T3
		R3(y)
		R3(z)
R1(x)		
W1(x)		
		W3(y)
		W3(z)
	R2(z)	
R1(y)		
W1(y)		
	R2(y)	
	W2(y)	
	R2(x)	
	W2(x)	

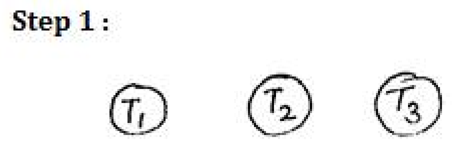
Solution :



$R_3(y)$ $R_3(z)$ $R_1(x)$ $\underset{1}{\underline{W_1(x)}}$ $\underset{3}{\underline{W_3(y)}}$ $\underset{4}{\underline{W_3(z)}}$ $\underset{f}{\underline{R_2(z)}}$

$\underset{e}{\underline{R_1(y)}}$ $\underset{2}{\underline{W_1(y)}}$ $\underset{d}{\underline{R_2(y)}}$ $\underset{c}{\underline{W_2(y)}}$ $\underset{b}{\underline{R_2(x)}}$ $\underset{a}{\underline{W_2(x)}}$

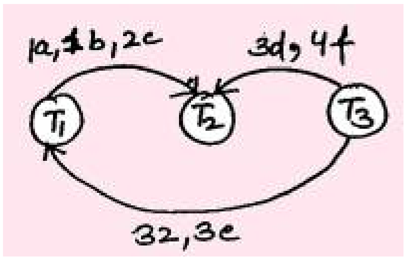
Step 1 : Create a node for each transaction.



Step 2 : Find the conflict pairs (RW, WR, WW) on same variable by different transactions.

W1(x); W2(x)
W1(x); R2(x)
W1(y); W2(y)
W3(y); R2(y)
W3(z); R2(z)
W3(y); W1(y)
W3(y); R1(y)

Step 3 : Draw an edge for each conflict pair.

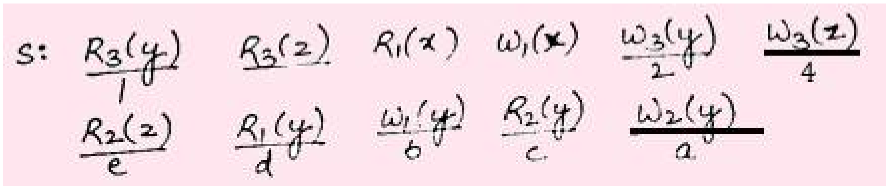


As no Cycle formed, therefore Schedule is conflict serializable schedule and serial schedule is T3 → T1 → T2.

Question 8 : Check whether the schedule is conflict serializable or not?

S: R3(y); R3(z); R1(x); W1(x); W3(y); W3(z); R2(z);
R1(y); W1(Y); R2(y); W2(y)

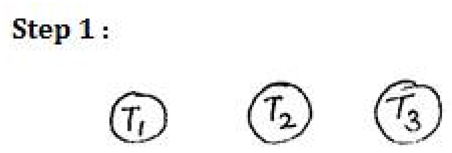
Solution :



S: $\underset{1}{\underline{R_3(y)}}$ $\underline{R_3(z)}$ $R_1(x)$ $W_1(x)$ $\underset{2}{\underline{W_3(y)}}$ $\underset{4}{\underline{W_3(z)}}$

$\underset{e}{\underline{R_2(z)}}$ $\underset{d}{\underline{R_1(y)}}$ $\underset{b}{\underline{W_1(y)}}$ $\underset{c}{\underline{R_2(y)}}$ $\underset{a}{\underline{W_2(y)}}$

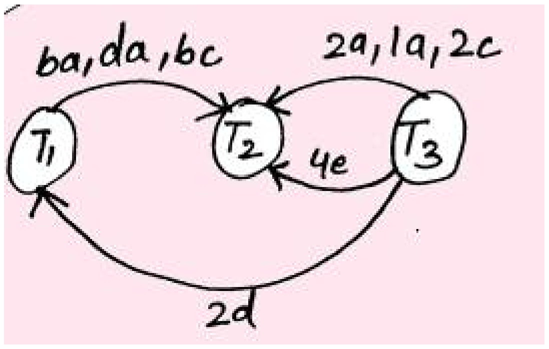
Step 1 : Create a node for each transaction.



Step 2 : Find the conflict pairs (RW, WR, WW) on same variable by different transactions.

W1(y); W2(y)
R1(y); W2(y)
W3(y); W2(y)
R3(y); W2(y)
W1(y); R2(y)
W3(y); R2(y)
W3(y); W1(y)
W3(z); R2(z)

Step 3 : Draw an edge for each conflict pair.

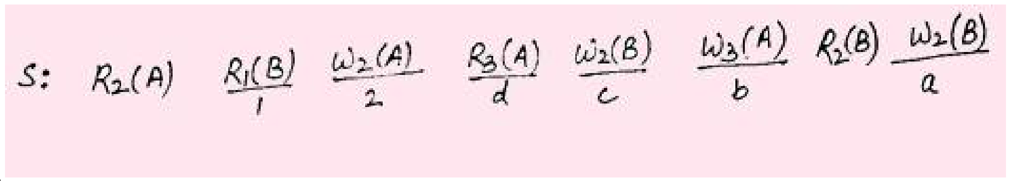


As Cycle formed, therefore Schedule is not a conflict serializable schedule.

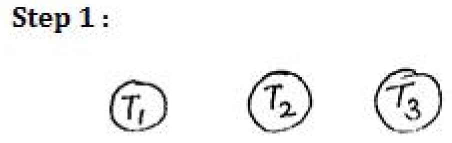
Question 9 : Check whether the schedule is conflict serializable or not?

S: R2(A); R1(B); W2(A); R3(A); W2(B); W3(A); R2(B); W2(B)

Solution :



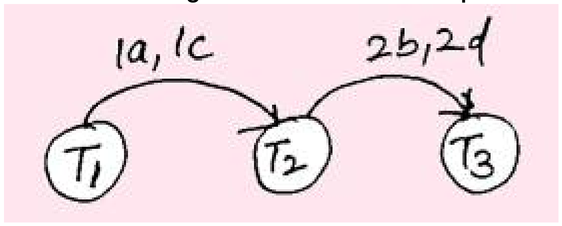
Step 1 : Create a node for each transaction.



Step 2 : Find the conflict pairs (RW, WR, WW) on same variable by different transactions.

R1(B); W2(B) [1a]
R1(B); W2(B) [1c]
W2(A); W3(A) [2b]
W2(A); R3(A) [2d]

Step 3 : Draw an edge for each conflict pair.



As No cycle formed, therefore Schedule is conflict serializable and serial schedule is T1 → T2 → T3