Transaction

- The transaction is a set of logically related operation. It contains a group of tasks.
- A transaction is an action or series of actions. It is performed by a single user to perform operations for accessing the contents of the database.

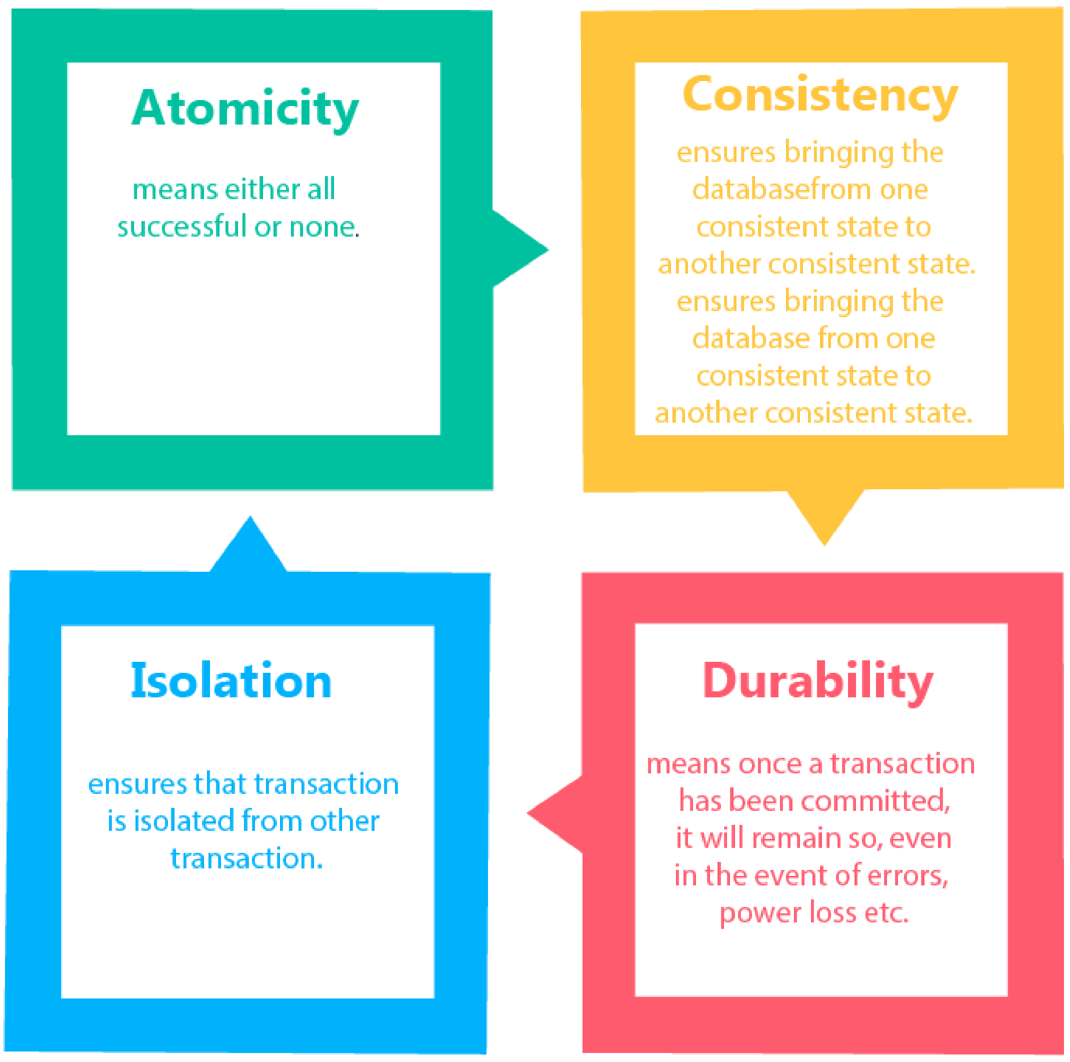
```
Ti :    read(A);

        A:= A-50;

        Write (A);

        Read(B);

        B:=B + 50;

        Write(B);
```

Transaction property

The transaction has the four properties. These are used to maintain consistency in a database, before and after the transaction.

Property of Transaction

1. Atomicity
2. Consistency
3. Isolation
4. Durability


Atomicity
means either all successful or none.
Consistency
ensures bringing the databasefrom one consistent state to another consistent state. ensures bringing the database from one consistent state to another consistent state.
Isolation
ensures that transaction is isolated from other transaction.
Durability
means once a transaction has been committed, it will remain so, even in the event of errors, power loss etc.

Atomicity

- It states that all operations of the transaction take place at once if not, the transaction is aborted.
- There is no midway, i.e., the transaction cannot occur partially. Each transaction is treated as one unit and either run to completion or is not executed at all.

Atomicity involves the following two operations:

Abort: If a transaction aborts then all the changes made are not visible.

Commit: If a transaction commits then all the changes made are visible.

Example: Let's assume that following transaction T consisting of T1 and T2. A consists of Rs 600 and B consists of Rs 300. Transfer Rs 100 from account A to account B.

T1	T2

Read(A)
A:= A-100
Write(A)
	Read(B)
	Y:= Y+100
	Write(B)

After completion of the transaction, A consists of Rs 500 and B consists of Rs 400.

If the transaction T fails after the completion of transaction T1 but before completion of transaction T2, then the amount will be deducted from A but not added to B. This shows the inconsistent database state. In order to ensure correctness of database state, the transaction must be executed in entirety.

Consistency

- The integrity constraints are maintained so that the database is consistent before and after the transaction.
- The execution of a transaction will leave a database in either its prior stable state or a new stable state.
- The consistent property of database states that every transaction sees a consistent database instance.
- The transaction is used to transform the database from one consistent state to another consistent state.

For example: The total amount must be maintained before or after the transaction.

1. Total before T occurs = 600+300=900
2. Total after T occurs= 500+400=900

Therefore, the database is consistent. In the case when T1 is completed but T2 fails, then inconsistency will occur.

Isolation

- It shows that the data which is used at the time of execution of a transaction cannot be used by the second transaction until the first one is completed.
- In isolation, if the transaction T1 is being executed and using the data item X, then that data item can't be accessed by any other transaction T2 until the transaction T1 ends.
- The concurrency control subsystem of the DBMS enforced the isolation property.

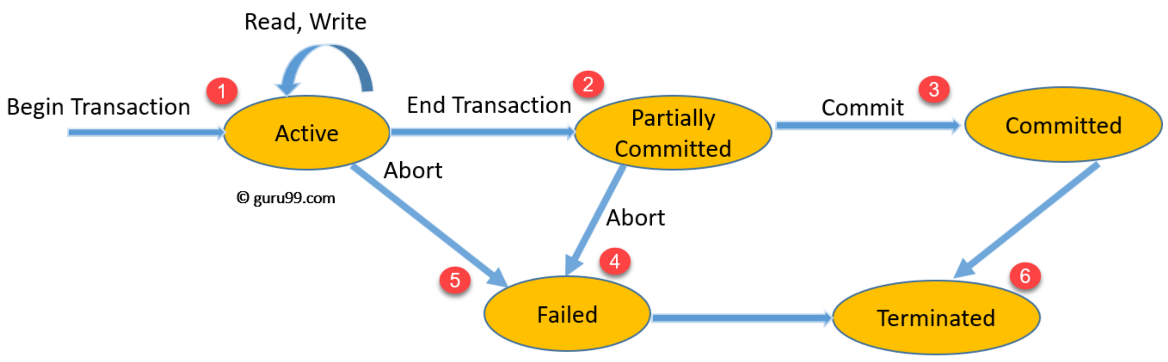
Durability

- The durability property is used to indicate the performance of the database's consistent state. It states that the transaction made the permanent changes.
- They cannot be lost by the erroneous operation of a faulty transaction or by the system failure. When a transaction is completed, then the database reaches a state known as the consistent state. That consistent state cannot be lost, even in the event of a system's failure.
- The recovery subsystem of the DBMS has the responsibility of Durability property.

States of Transactions

The various states of a Database Transaction are listed below

State	**Transaction types**
Active State	A transaction enters into an active state when the execution process begins. During this state read or write operations can be performed.
Partially Committed	A transaction goes into the partially committed state after the end of a transaction.
Committed State	When the transaction is committed to state, it has already completed its execution successfully. Moreover, all of its changes are recorded to the database permanently.
Failed State	A transaction considers failed when any one of the checks fails or if the transaction is aborted while it is in the active state.
Terminated State	State of transaction reaches terminated state when certain transactions which are leaving the system can't be restarted.



State Transition Diagram for a Database Transaction

1. Once a transaction states execution, it becomes active. It can issue READ or WRITE operation.
2. Once the READ and WRITE operations complete, the transactions becomes partially committed state.
3. Next, some recovery protocols need to ensure that a system failure will not result in an inability to record changes in the transaction permanently. If this check is a success, the transaction commits and enters into the committed state.
4. If the check is a fail, the transaction goes to the Failed state.
5. If the transaction is aborted while it's in the active state, it goes to the failed state. The transaction should be rolled back to undo the effect of its write operations on the database.
6. The terminated state refers to the transaction leaving the system.

- Restart
- Kill

CONCURRENT EXECUTION IN TRANSACTION

Transaction-processing systems usually allow multiple transactions to run concurrently. Allowing multiple transactions to update data concurrently causes several complications with consistency of the data.

There are two good reasons for allowing concurrency:

- Improved throughput and resource utilization
- Reduced waiting time

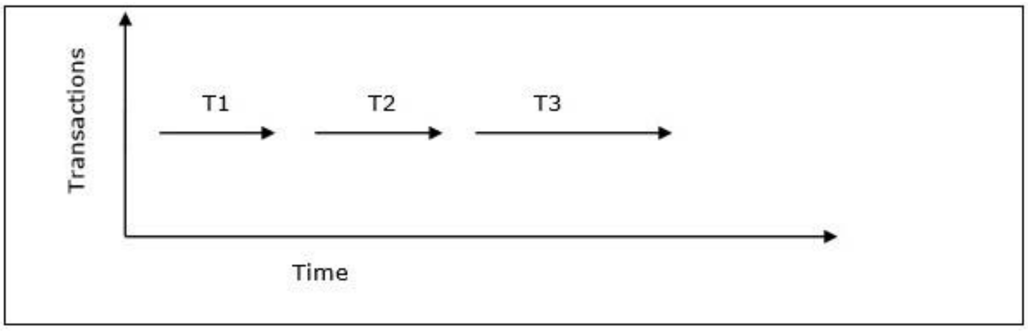
Schedules and Conflicts

In a system with a number of simultaneous transactions, a **schedule** is the total order of execution of operations. Given a schedule S comprising of n transactions, say T1, T2, T3………..Tn; for any transaction Ti, the operations in Ti must execute as laid down in the schedule S.

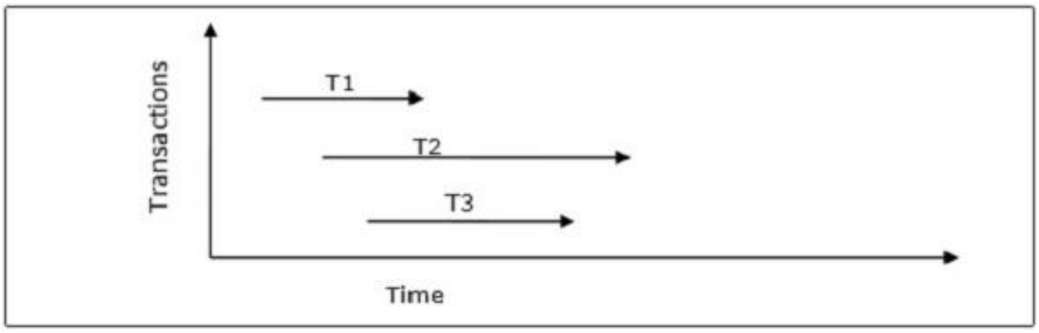
Types of Schedules

There are two types of schedules −

- **Serial Schedules** − In a serial schedule, at any point of time, only one transaction is active, i.e. there is no overlapping of transactions. This is depicted in the following graph −



- **Parallel Schedules** − In parallel schedules, more than one transactions are active simultaneously, i.e. the transactions contain operations that overlap at time. This is depicted in the following graph −



Conflicts in Schedules

In a schedule comprising of multiple transactions, a **conflict** occurs when two active transactions perform non-compatible operations. Two operations are said to be in conflict, when all of the following three conditions exists simultaneously −

- The two operations are parts of different transactions.
- Both the operations access the same data item.
- At least one of the operations is a write_item() operation, i.e. it tries to modify the data item.

Cases:

Case 1: R(X) of T1 and R(X) of T2 are non-conflicting operations because none of them is write operation.

Case 2: Operation W(X) of transaction T1 and operation R(X) of transaction T2 are conflicting operations.

Case 3: Operation R(X) of transaction T1 and operation W(X) of transaction T2 are conflicting operations.

Case 4: Operations W(X) of T1 and W(X) of T2 are conflicting operations.

Scheduling in DBMS is majorly classified into

- Serial and
- on-Serial Schedules.

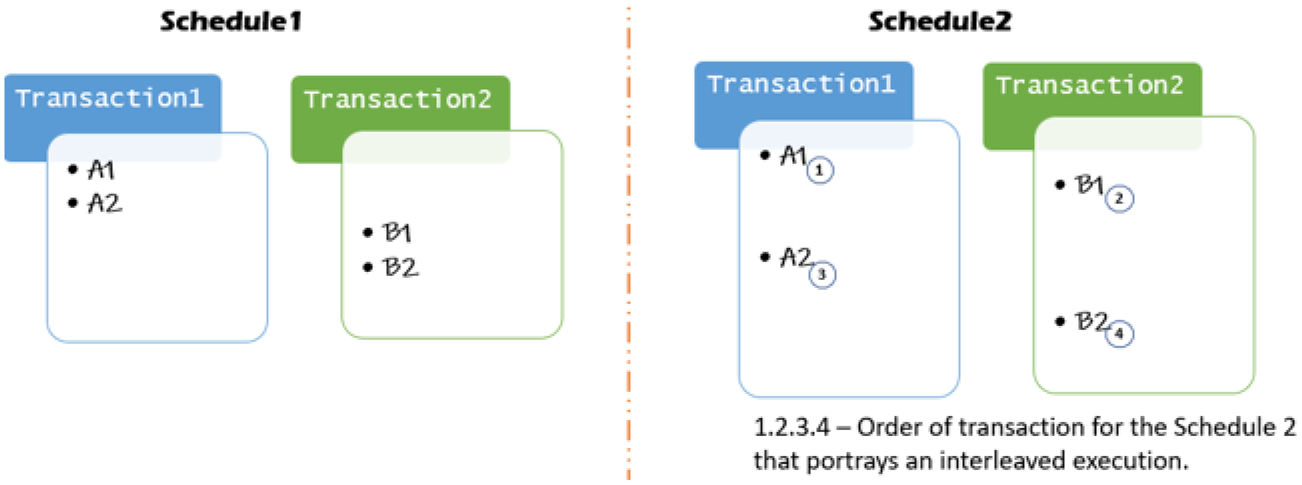
As a part of transaction management, it is important to verify if a non-serial schedule is serializable especially when the transactions have concurrent execution. A given non-serial schedule of 'n' Transactions is said to be serializable if there exists some kind of equivalent serial schedule to the same 'n' transactions.

What is Serializability in DBMS?

Serial schedule both by definition and execution means that the transactions bestowed upon it will take place serially, that is, one after the other. This leaves no place for inconsistency within the database. But, when a set of transactions are scheduled non-serially, they are interleaved leading to the problem of concurrency within the database. Non-serial schedules do not wait for one transaction to complete for the other one to begin. Serializability in DBMS decides if an interleaved non-serial schedule is serializable or not.

Example of Serializability

Consider 2 schedules, Schedule1 and Schedule2:


Schedule1
Transaction1
A1
A2
Transaction2
B1
B2
Schedule2
Transaction1
A1
1
A2
3
Transaction2
B1
2
B2
4
1.2.3.4 – Order of transaction for the Schedule 2 that portrays an interleaved execution.

- Schedule1 is a serial schedule consisting of Transaction1 and Transaction2 wherein the operations on data item A (A1 and A2) are performed first and later the operations on data item B (B1 and B2) are carried out serially.
- Schedule2 is a non-serial schedule consisting of Transaction1 and Transaction2 wherein the operations are interleaved.

Explanation: In the given scenario, schedule2 is serializable if the output obtained from both Schedule2 and Schedule1 are equivalent to one another In a nutshell, a transaction within a given non-serial schedule is serializable if its outcome is equivalent to the outcome of the same transaction when executed serially.

Types of Serializability

A schedule can be checked for serializability in one of the 3 methods

mentioned below:

1. Result Equivalent Schedule

- Two schedules, S1 and S2 are said to result equivalent if they produce the same output obtained when the schedules are serially executed.
- Often, this kind of schedule is given the least significance since the result derived are mainly focussed on the output which in some cases may vary for the same set of inputs or might produce the same output for a different set of inputs.

2. Conflict Equivalent Schedule

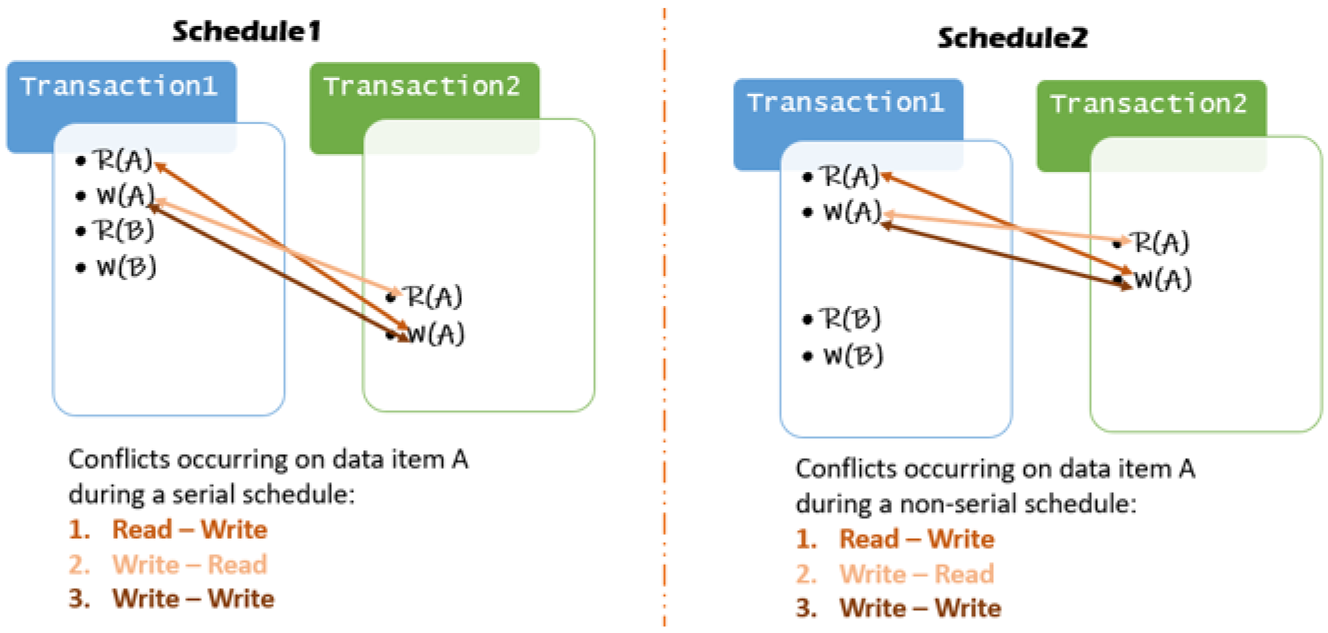
When either of a conflict operation such as Read-Write or Write-Read or Write-Write is implemented on the same data item at the same time within different transactions then the schedule holding such transactions is said to be a conflict schedule. The prerequisites for such conflict schedule are:

1. The conflict operations are to be implemented on the same data item.
2. The conflict operations (RW, WR, WW) must take place within different transactions.
3. At least one of the conflict operations must be the write operation.
4. Two Read operations will not create any conflict.

Two schedules (one being serial schedule and another being non-serial) are said to be conflict serializable if the conflict operations in both the schedules are executed in the same order.

Example:

Consider 2 schedules, Schedule1 and Schedule2,

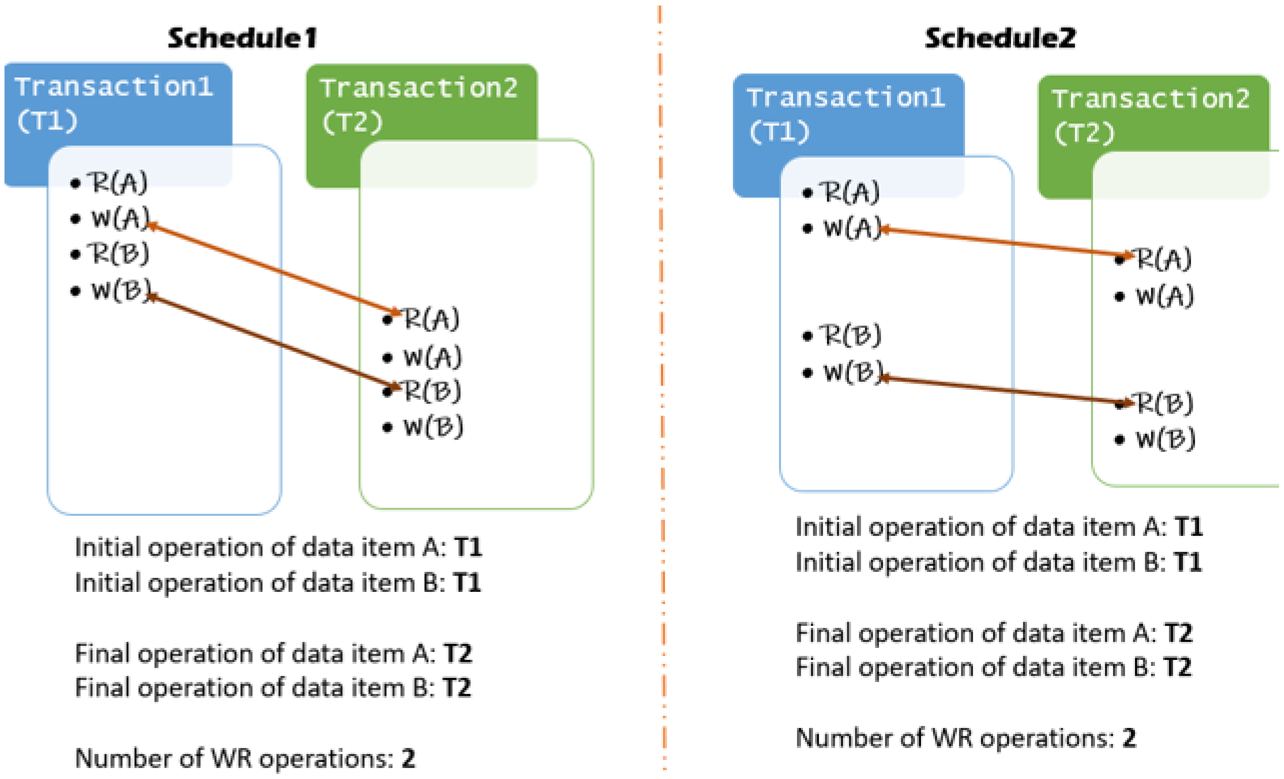


Schedule2 (a non-serial schedule) is considered to be conflict serializable when its conflict operations are the same as that of Shedule1 (a serial schedule).

3. View Equivalent Schedule

Two schedules (one being serial schedule and another being non-serial) are said to be view serializable if they satisfy the rules for being view equivalent to one another.

The rules to be upheld are:

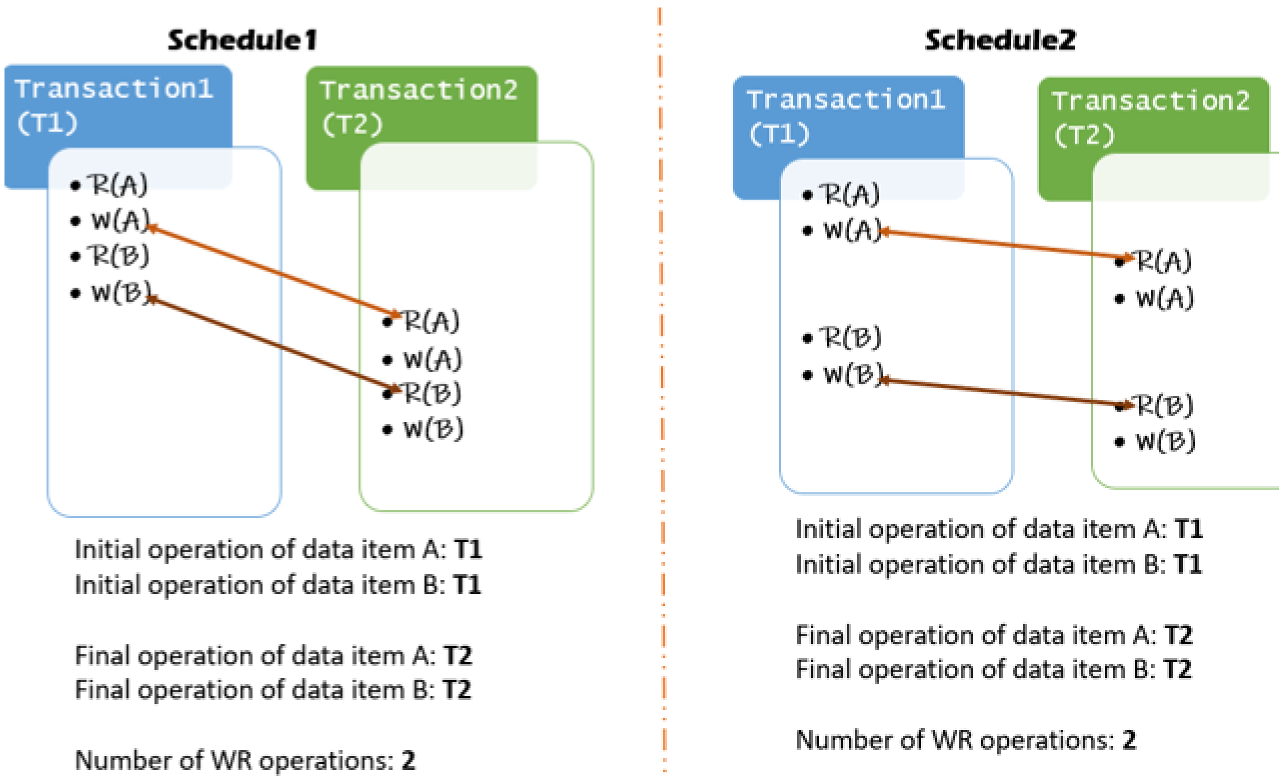

Schedule1
Transaction1 (T1)
Transaction2 (T2)
R(A)
W(A)
R(B)
W(B)
R(A)
W(A)
R(B)
W(B)
Initial operation of data item A: T1
Initial operation of data item B: T1
Final operation of data item A: T2
Final operation of data item B: T2
Number of WR operations: 2
Schedule2
Transaction1 (T1)
Transaction2 (T2)
R(A)
W(A)
R(A)
W(A)
R(B)
W(B)
R(B)
W(B)
Initial operation of data item A: T1
Initial operation of data item B: T1
Final operation of data item A: T2
Final operation of data item B: T2
Number of WR operations: 2

1. Initial values of the data items involved within a schedule must be the same.
2. Final values of the data items involved within a schedule must be the same.

3. The number of WR operations performed must be equivalent for the schedules involved.

Example:

Consider 2 schedules, Schedule1 and Schedule2:


Schedule1
Transaction1 (T1)
Transaction2 (T2)
• R(A)
• W(A)
• R(B)
• W(B)
• R(A)
• W(A)
• R(B)
• W(B)
Initial operation of data item A: T1
Initial operation of data item B: T1
Final operation of data item A: T2
Final operation of data item B: T2
Number of WR operations: 2
Schedule2
Transaction1 (T1)
Transaction2 (T2)
• R(A)
• W(A)
• R(A)
• W(A)
• R(B)
• W(B)
• R(B)
• W(B)
Initial operation of data item A: T1
Initial operation of data item B: T1
Final operation of data item A: T2
Final operation of data item B: T2
Number of WR operations: 2

The (non-serial) Schedule2 is considered as a view equivalent of the (serial) Schedule1, when the 3 rules of view serializability are satisfied. For the example shown above,

- The Initial transaction of read operation on the data items A and B both begin at T1

- The Final transaction of write operations on the data items A and B both end at T2
- The number of updates from write-read operations are 2 in both the cases

Hence satisfying all the rules required, Schedule2 becomes view serializable w.r.t Schedule1.

Benefits of Serializability in DBMS

Let’s first understand the difference between a serial and non-serial schedule for a better understanding of the benefits that serializability provides. In the case of a serial schedule, the multiple transactions involved are executed one after the other sequentially with no overlap. This helps maintain the consistency in the database but limits the scope of concurrency and often a smaller transaction might end up waiting for a long time due to an execution of a previous longer transaction. Serial schedule also consumes a lot of CPU resources which gets wasted due to the serial execution.

In the case with a non-serial schedule, the multiple transactions executed are interleaved leading to inconsistency in the database but at the same time helps overcome the disadvantages of a serial schedule such as concurrent execution and wastage of CPU resources.

It’s established that the execution of multiple transactions in a non-serial schedule takes place concurrently. And because of the multiple combinations involved, the output obtained may be incorrect at times which cannot be afforded. This is where serializability comes into the picture and help us determine if the output obtained from a parallelly executed schedule is correct or not.

In other words, Serializability serves as a measure of correctness for the transactions executed concurrently. It serves a major role in concurrency control that is crucial for the database and is considered to provide maximum isolation between the multiple transactions involved. The process of Serializability can also help in achieving the database consistency which otherwise is not possible for a non-serial schedule.

Conclusion

- Serializable means obtaining an equivalent output as of a serial schedule for the same ‘n’ number of transactions.

- Serializability helps preserve the consistency and concurrency of a database.
- There are 2 methods widely used to check serializability i.e. Conflict equivalent and View equivalent.
- As a rule of thumb, it can be stated that all conflict serializable schedules can be view serializable, but all view serializable schedules may or may not be conflict serializable.