A transaction can be defined as a group of tasks. A single task is the minimum processing unit which cannot be divided further.

Let's take an example of a simple transaction. Suppose a bank employee transfers Rs 500 from A's account to B's account. This very simple and small transaction involves several low-level tasks.

**A's Account**

```
Open_Account(A)
Old_Balance = A.balance
New_Balance = Old_Balance - 500
A.balance = New_Balance
Close_Account(A)
```

**B's Account**

```
Open_Account(B)
Old_Balance = B.balance
New_Balance = Old_Balance + 500
B.balance = New_Balance
Close_Account(B)
```

## Acid properties

A transaction is a very small unit of a program and it may contain several lowlevel tasks. A transaction in a database system must maintain Atomicity, Consistency, Isolation, and Durability − commonly known as ACID properties − in order to ensure accuracy, completeness, and data integrity.

**Atomicity** − This property states that a transaction must be treated as an atomic unit, that is, either all of its operations are executed or none. There must be no state in a database where a transaction is left partially completed. States should be defined either before the execution of the transaction or after the execution/abortion/failure of the transaction.

**Consistency** − The database must remain in a consistent state after any transaction. No transaction should have any adverse effect on the data residing in the database. If the database was in a consistent state before the execution of a transaction, it must remain consistent after the execution of the transaction as well.

**Durability** − The database should be durable enough to hold all its latest updates even if the system fails or restarts. If a transaction updates a chunk of data in a database and commits, then the database will hold the modified data. If a transaction commits but the system fails before the data could be written on to the disk, then that data will be updated once the system springs back into action.

**Isolation** − In a database system where more than one transaction are being

executed simultaneously and in parallel, the property of isolation states that all the transactions will be carried out and executed as if it is the only transaction in the system. No transaction will affect the existence of any other transaction.

## Serializability

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

**Schedule** − A chronological execution sequence of a transaction is called a schedule. A schedule can have many transactions in it, each comprising of a number of instructions/tasks.

**Serial Schedule** − It is a schedule in which transactions are aligned in such a way that one transaction is executed first. 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 a serial schedule, as transactions are executed in a serial manner.

In a multi-transaction environment, serial schedules are considered as a benchmark. The execution sequence of an instruction in a transaction cannot be changed, but two transactions can have their instructions executed in a random fashion. This execution does no harm if two transactions are mutually independent and working on different segments of data; but in case these two transactions are working on the same data, then the results may vary. This ever-varying result may bring the database to an inconsistent state.

To resolve this problem, we allow parallel execution of a transaction schedule, if its transactions are either serializable or have some equivalence relation among them.

## Equivalence Schedules

An equivalence schedule can be of the following types-

### Result Equivalence

If two schedules produce the same result after execution, they are said to be result equivalent. They may yield the same result for some value and different results for another set of values. That's why this equivalence is not generally considered significant.

### View Equivalence

Two schedules would be view equivalence if the transactions in both the schedules perform similar actions in a similar manner.

### For example −

+If T reads the initial data in S1, then it also reads the initial data in S2.

+If T reads the value written by J in S1, then it also reads the value written by J in S2.

+If T performs the final write on the data value in S1, then it also performs the final write on the data value in S2.

** Conflict Equivalence** Two schedules would be conflicting if they have the following properties −

- Both belong to separate transactions.

- Both accesses the same data item.

- At least one of them is "write" operation. Two schedules having multiple transactions with conflicting operations are said to be conflict equivalent if and only if −

- Both the schedules contain the same set of Transactions.

- The order of conflicting pairs of operation is maintained in both the schedules. ** Note** − View equivalent schedules are view serializable and conflict equivalent schedules are conflict serializable. All conflict serializable schedules are view serializable too.

**States of Transactions** A transaction in a database can be in one of the following states −

Transaction States

**Active** − In this state, the transaction is being executed. This is the initial state of every transaction.

**Partially Committed** − When a transaction executes its final operation, it is said to be in a partially committed state.

**Failed** − A transaction is said to be in a failed state if any of the checks made by the database recovery system fails. A failed transaction can no longer proceed further.

**Aborted** − If any of the checks fails and the transaction has reached a failed state, then the recovery manager rolls back all its write operations on the database to bring the database back to its original state where it was prior to the execution of the transaction. Transactions in this state are called aborted. The database recovery module can select one of the two operations after a transaction aborts −

- Re-start the transaction
- Kill the transaction

**Committed** − If a transaction executes all its operations successfully, it is said to be committed. All its effects are now permanently established on the database system.