DBMS Transaction

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

An example of transaction can be bank accounts of two users, say A & B. When a bank employee transfers amount of Rs. 500 from A's account to B's account, a number of tasks are executed behind the screen. This very simple and small transaction includes several steps: decrease A's bank account from 500

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

In simple words, the transaction involves many tasks, such as opening the account of A, reading the old balance, decreasing the 500 from it, saving new balance to account of A and finally closing it. To add amount 500 in B's account same sort of tasks need to be done:

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

A simple transaction of moving an amount of 500 from A to B involves many low level tasks.

ACID Properties

A transaction may contain several low level tasks and further a transaction is very small unit of any program. A transaction in a database system must maintain some properties in order to ensure the accuracy of its completeness and data integrity. These properties are refer to as ACID properties and are mentioned below:

- **Atomicity:** Though a transaction involves several low level operations but 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 database where the 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:** This property states that after the transaction is finished, its database must remain in a consistent state. There must not be any possibility that some data is incorrectly affected by the execution of transaction. If the database was in a consistent state before the execution of the transaction, it must remain in consistent state after the execution of the transaction.
- **Durability:** This property states that in any case all updates made on the database will persist even if the system fails and restarts. If a transaction writes or updates some data in database and commits that data will always be there in the database. If the transaction commits but data is not written on the disk and the system fails, that data will be updated once the system comes up.
- **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 more than one transaction is 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 transaction is called schedule. A schedule can have many transactions in it, each comprising of number of instructions/tasks.

- **Serial Schedule:** 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 next transaction is executed. Transactions are ordered one after other. This type of schedule is called serial schedule as transactions are executed in a serial manner.

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

  To resolve the problem, we allow parallel execution of transaction schedule if transactions in it are either serializable or have some equivalence relation between or among transactions.

  Equivalence schedules: Schedules can equivalence of the following types:

- **Result Equivalence:**

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

- **View Equivalence:**

  Two schedules are view equivalence if transactions in both schedules perform similar actions in similar manner.

  For example:

  - If T reads initial data in S1 then T also reads initial data in S2
  - If T reads value written by J in S1 then T also reads value written by J in S2
  - If T performs final write on data value in S1 then T also performs final write on data value in S2

- **Conflict Equivalence:**

  Two operations are said to 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

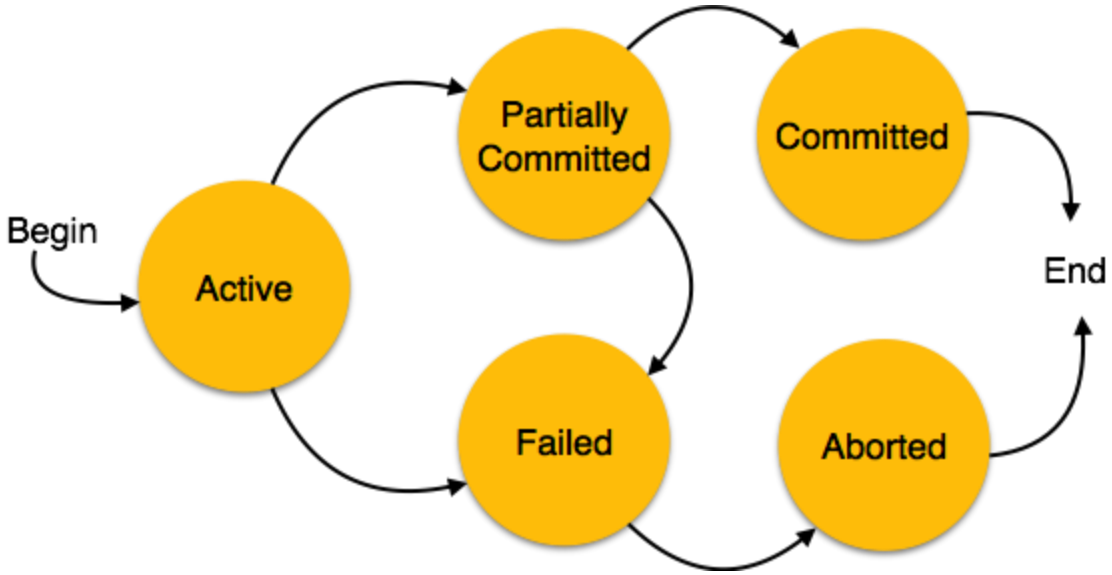
  Two schedules have more than one transactions with conflicting operations are said to be conflict equivalent if and only if:

  - Both schedules contain same set of Transactions
  - The order of conflicting pairs of operation is maintained in both schedules

  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 state:



[*Image: 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 this state. After execution of all operations, the database system performs some checks e.g. the consistency state of database after applying output of transaction onto the database.
- **Failed:** If any checks made by database recovery system fails, the transaction is said to be in failed state, from where it can no longer proceed further.
- **Aborted:** If any of checks fails and transaction reached in Failed state, the recovery manager rolls back all its write operation on the database to make database in the state where it was prior to start of execution of transaction. Transactions in this state are called aborted. Database recovery module can select one of the two operations after a transaction aborts:

  - Re-start the transaction
  - Kill the transaction

- **Committed:** If transaction executes all its operations successfully it is said to be committed. All its effects are now permanently made on database system.

Source:

http://www.tutorialspoint.com/dbms/dbms_transaction.htm