TRANSACTION CONCEPTS

A transaction is a collection of operations that forms single logical unit of work.

Simple Transaction Example

1. Read your account balance
2. Deduct the amount from your balance
3. Write the remaining balance to your account
4. Read your friend‘s account balance
5. Add the amount to his account balance
6. Write the new updated balance to his account

This whole set of operations can be called a transaction

Transaction processing system

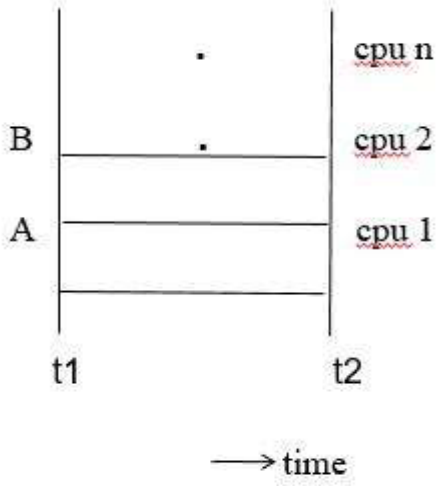
– The system with large database and hundreds of concurrent users that are executing database transaction.

– Eg :reservation system , banking system etc

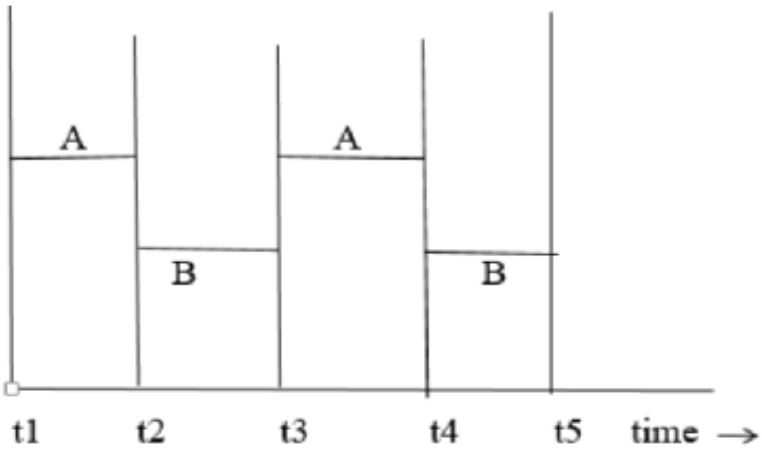
(i) Concurrent access

- Mutiple user accessing a system at the same time
- Single user-one user at a time can use a system
- Multi user-many user use the system at a time.
- It can be achieved by multiprogramming:

(ii) Parallel- multi-users access different resources at the same time.



(iii) Interleaved- Multiple users access a single resource based on time



Transaction access data using two operations

- **Read(x)**

  It transfer the data item **x** from the database to a local buffer belonging to the transaction that executed the read operation.

- **Write(x)**

  It transfer the data item **x** from the local buffer of the transaction to the database i.e. it write back to the database

ACID Properties

To ensure the integrity of data during a transaction, the database system maintains the following properties. These properties are widely known as ACID properties:

(i) 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.

(ii) 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.

(iii) 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.

(iv) 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.

Example : Example of Fund Transfer

transaction to transfer $50 from account A to account B:

1. **read**(*A*)
2. *A* := *A* – 50
3. **write**(*A*)
4. **read**(*B*)
5. *B* := *B* + 50
6. **write**(*B)*

Atomicity requirement

– if the transaction fails after step 3 and before step 6, money will be ―lost‖ leading to an inconsistent database state

– the system should ensure that updates of a partially executed transaction are not reflected in the database

Durability requirement

– once the user has been notified that the transaction has completed, the updates to the database by the transaction must persist even if there are software or hardware failures.

Isolation requirement

― if between steps 3 and 6, another transaction T2 is allowed to access the partially updated database, it will see an inconsistent database **T1 T2**

1. **read**(*A*)

2. *A* := *A* – 50

3. **write**(*A*)

#read(A), read(B), print(A+B)

4. **read**(*B*)

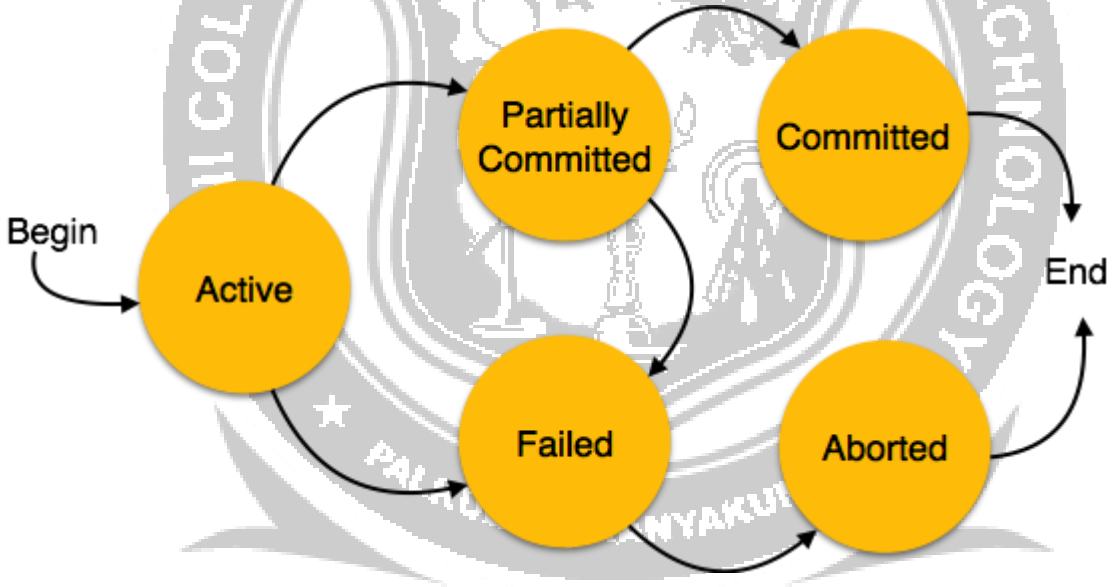
5. *B* := *B* + 50

6. **write**(*B)*

Isolation can be ensured trivially by running transactions **serially**– that is, one after the other.

States of Transactions

A transaction in a database can be in one of the following 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.