Transaction System

Transaction is a logical unit of work that represents the real-world events.

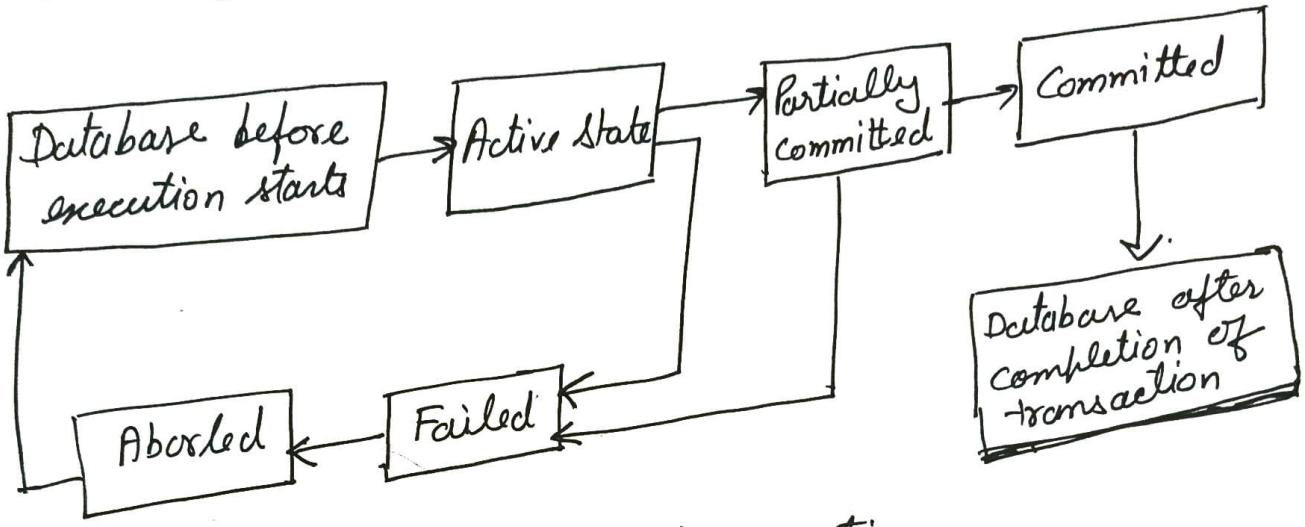
A transaction is also defined as any one execution of a user program in a Database Management System (DBMS).

Transaction management is the ability of a database management to manage the different transaction that occur within it.

During transactions data items can be read or updated or both.

Transaction States

A transaction must be in one of the following states as below:



States of the transaction:

1. Active state: It is the Initial state of transaction. During execution of statements, a transaction is in active state.
2. Partially committed: A transaction is in partially committed state, when all the statements within transaction are executed but transaction is not committed.
3. Failed: In any case, if transaction can not be proceeded further then transaction is in failed state.
4. Committed: After successful completion of transaction, it is in committed state.

ACID Properties of Transaction

A database system is required to maintain the following properties of transactions to ensure the integrity of the data

1. **Atomicity**: A transaction must be atomic. Atomic transaction means either all operations within a transaction are completed or none.

2. **Consistency**: A transaction must be executed in isolation. It means variables used by a transaction can not be changed by any other transaction concurrently.

3. **Isolation**: During concurrent tranction each transaction must be unaware of other transaction. For any transaction T_i, it appears to T_i that any other transaction T_k is either finished before starting of T_i or started after T_i finished.

4. **Durability**:— changes are made permanent to database after succesful completion of transaction even in the case of system feilure or crash.

Serializability.

At the heart of any concurrency control scheme is the concept of serializability.

A transaction T essentially consist of a sequence (or ordering) of read and write operations on the data items followed by the commit or abort operation (but not both). Moreover there can not be simultaneous read or write operations on the same data items. In a transaction no read or write operation can occur after the transaction has issued a commit or abort.

Serializability -

In serializability, ordering of read/writes is important:

(a) If two transactions only read a data item, they do not conflict and order is not important.

(b) If two transactions either read or write completely separate data items, they do not conflict and order is not important.

(c) If one transaction writes a data item and another reads or writes same data item, order of execution is important.

View Serializability:

offers less stringent definition of schedule equivalence than conflict serializability.

Two schedules S_1 & S_2 are view equivalent if:

- for each data item x, if T_i reads initial value of x in S_1, T_i must also read initial value of x in S_2.
- for each read on x by T_i in S_1, if value read by x is written by T_j, T_i must also read value of x produced by T_j in S_2.
- for each data item x, if last write on x performed by T_i in S_1, same transaction must perform final write on x in S_2.

* Schedule is view serialisable if it is view equivalent to a serial schedule.
* Every conflict serializable schedule is view serializable, although converse is not true.

Conflict Serializable schedule orders any conflicting operations in same way as some serial execution.

we can use precedence graph to test for serializability.

If the precedence graph contains cycle schedule is not conflict serializable.

T_1

begin_Transaction

read (balx)

balx = balx + 100

write (balx)

read (baly)

baly = baly - 100

write (baly)

commit

T_2

begin_transaction

read (balx)

balx = balx * 1.1

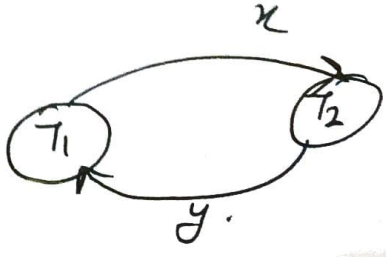
write (balx)

read (baly)

baly = baly * 1.1

write (baly)

commit



Recoverability.

If a transaction T_i fails, for whatever reason. we need to undo (rollback) the effect of this transaction to ensure the atomicity property of the transaction.

In a system that allows concurrents execution it is necessary also to ensure that any transaction T_j that is dependent on T_i (ie, T_j has read data written by T_i) is also aborted. To achieve this surety we need to place restrictions on the type of schedules permitted in the system.

A recovery process is an integral part of a database system, which is responsible for detection of failures and recovery of database. There are two types of failures —

1. Loss of volatile storage
2. Loss of non-volatile storage
3. cascading Rollback.

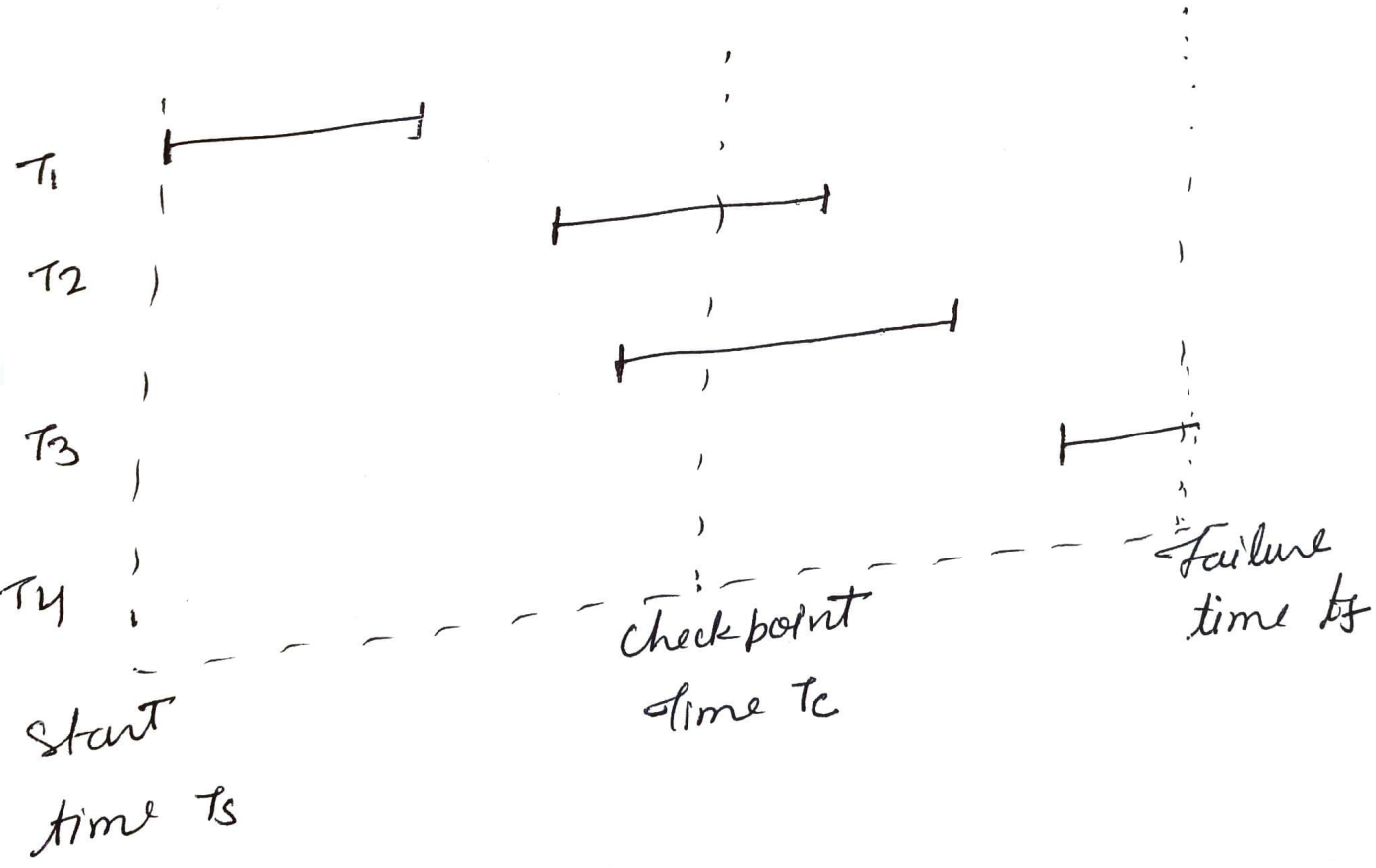
Log based recovery.

- The very important structure is used for recording database modification is the log.
- The log is a sequence of log Records, recording all the updates activities in the database.
- There are many types of log records, An update log record has these fields —
  - → Transaction identifier
  - → Data-item identifiers
  - → old value
  - → New value
- There log records are —
  - $\langle T_i \text{ start} \rangle$ Transaction T_i started
  - $\langle T_i, X_j, V_1, V_2 \rangle$ Transaction T_i has performed a write on data item X_j. X_j has value V_1 before the write & will have value V_2 after the write.
  - $\langle T_i \text{ commit} \rangle$ transaction T_i has committed.
  - $\langle T_i \text{ abort} \rangle$ transaction T_i has aborted.

Checkpoints, Deadlock

A checkpoint is a point of synchronization between the database and the transaction log file. All points are forced written to secondary storage at the check point. checkpoints also called syncpoints or save point.

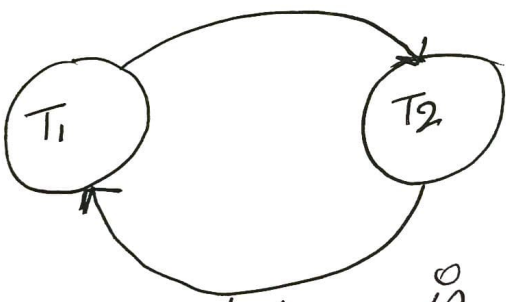
We used checkpoints to the number of log records that the system must scan when it recovers from a crash.



Deadlock

consider the following example:

T_1	T_2
read_lock (Y);	
read_item (Y);	read_lock (X);
	read_item (X);
write_lock (X)	
	write_lock Y



Deadlock: Deadlock is a situation where a process or set of process are blocked, waiting on an event which will never occur. We can say "A state where neither of transactions can ever proceed with its normal execution. This situation is called deadlock".

Approaches to deal with deadlocks.

1. Deadlock prevention techniques (DPT)
2. Deadlock detection protocals (DDP)
3. Recovery from deadlock

1. Deadlock Prevention techniques (DPT)

- Using lock-based protocal
  eg. 2PL
- Using wait-die
- Using wound-wait.

2- Deadlock detection: "A deadlock exists in a system if and only if the wait-for graph contain a cycle." Each transaction involved in the cycle is said to be deadlock-

3. Recovery from deadlock: when a detection algorithm determines that a deadlock exists the system must recover from the deadlock. There are three actions need to be taken —

(i) Selection of a victim
(ii) Rollback
(iii) Starvation

Distributed Data Storage.

- Assume relational data model

- Replication
  System maintains multiple copies of data, stored in different sites, for faster retrieval and fault tolerance.

- Fragmentation
  - Relation is partitioned into several fragments stored in distinct sites.

- Replication and fragmentation can be combined
  - Relation is partitioned into several fragments: System maintains several identical replicas of each such fragment.

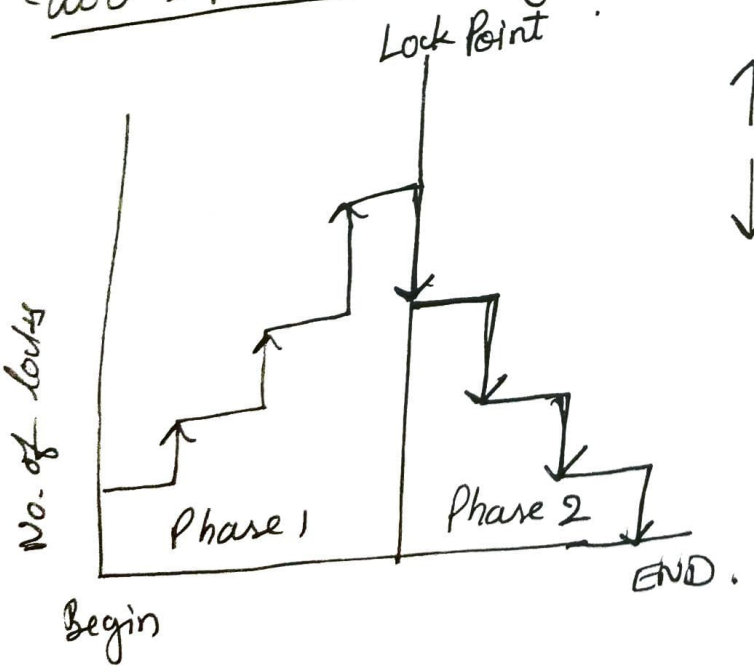
- A relation or fragment of a relation is replicated if it is stored redundantly in two or more sites.
  - Full replication of a relation is the case where the relation is stored at all sites.

Distributed database Management System (DDBMS)

→ A collection of multiple, logically interrelated databases distributed over a computer network. A distributed database management system is as the software system that permits the management of the distributed database and make the distribution transparent to the users.

Concurrency control Algorithms:

Two-Phase locking:



↑ Obtain lock

↓ Release lock.