TRANSACTIONS

Transaction

A transaction is a unit of program execution that accesses and possibly updates various data items.

A transaction is delimited by the statements begin transaction and end transaction.

To ensure integrity of the data, we require that the database system maintain the following properties of the transactions.

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

Atomicity :-

Either all operations of the transaction are reflected properly in the database or none are.

Consistency :-

Execution of a transaction in isolation (i.e. with no other transaction executing concurrently) preserves the consistency of the database.

Isolation :-

Even though multiple transactions may execute concurrently, the system guarantees that, for every

pair of transactions T_i and T_j, it appears to T_i that T_j finished execution before T_i started or T_j started execution after T_i finished. Thus, each transaction is unaware of other transactions executing concurrently in the system.

Durability :-

After a transaction completes successfully, the changes it has made to the database persist, even if there are system failures.

These properties are often called ACID properties The acronym is derived from the first letter of each of the four properties.

Atomicity → A

Consistency → C

Isolation → I

Durability → D

Transactions access data using two operations

1. Read(x)

2. Write(x)

Read(X)

It transfers the data item X from the database to a local buffer belonging to the transaction that executed the read operation.

Write(X)

It transfers the data item X from the local buffer of the transaction that executed the write back to the database.

* Ensuring atomicity is the responsibility of the transaction management component.

* Ensuring consistency is the responsibility of the application programmer who codes the transaction.

* Ensuring Isolation property is the responsibility of concurrency control component.

* Ensuring durability is the responsibility of recovery-management component.

Transaction States :-

A transaction must be in one of the following states

Active :-

It is the initial state. The transaction stays in this state while it is executing

partially committed :-

The transaction is said to be in partially committed state after the final statement has been executed

failed

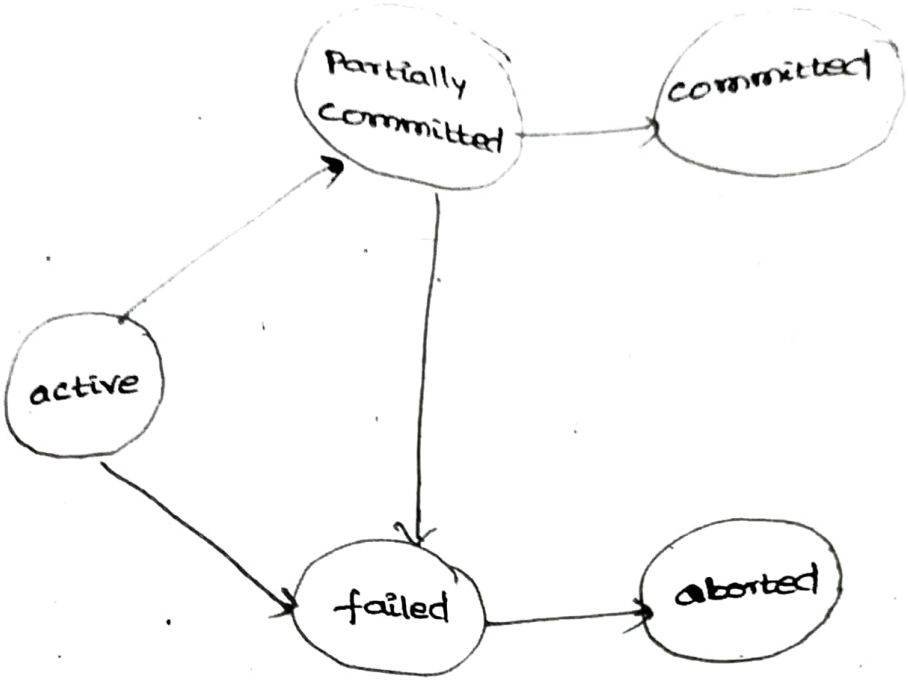
The transaction is said to be in failed state after the discovery that normal execution can no longer proceed.

Aborted

The transaction is said to be in aborted state after the transaction has been rolled back and the database has been restored to its state prior to the start of the transaction.

commited

The transaction is said to be in committed state after successful completion of the transaction.



State diagram of a transaction.

Implementation of Atomicity and Durability

The recovery management component of a database system can support atomicity and durability through a technique known as shadow-copy technique.

Shadow-copy technique

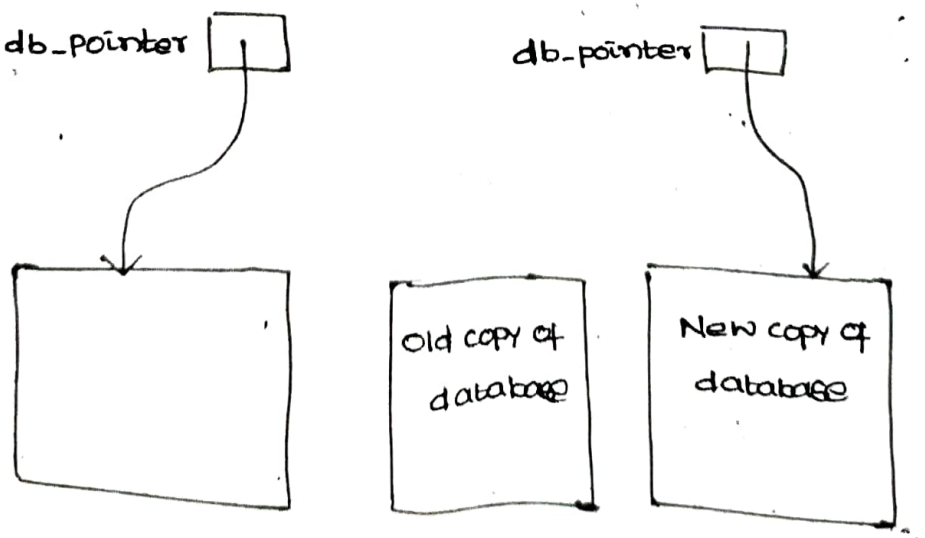
* It is based on making copies of the database, called shadow copies, assumes that only one transaction is active at a time.

* It assumes that the database is a file on disk.

* A pointer called db-pointer is maintained on disk; it points to the current copy of the database.

In this technique, a transaction that wants to update the database, first creates a complete copy of the database. All updates are done on the new database copy, leaving the original copy, (shadow copy) untouched. If at any point, the transaction has to be aborted, the system merely deletes the new copy. The old copy of the database has not been affected.

If the transaction is committed, the operating system is asked to make sure that all pages of the new copy of the database have been written out to disk. After the operating system has written all the pages to disk, the database system updates the pointer to point to the new copy of the database; the new copy then becomes the current copy of the database. The old copy of the database is then deleted.



(a) Before Update

(b) After Update.

Concurrent transactions

Transaction-processing systems allow multiple transactions to run concurrently.

Advantages of concurrency.

* Improved throughput and resource utilization
* Reduced waiting time.

Schedule

A schedule is a collection of transactions and represent the chronological order in which the instructions are executed in the system.

Schedules are of two types.

1. Serial schedule.
2. Concurrent schedule.

Serial Schedule

A schedule is said to be serial if one transaction is followed by other.

Concurrent Schedule.

A schedule is said to be concurrent if the transactions involved in the schedule are executed concurrently.

Let T_1 be a transaction which transfers an amount of Rs. 100 from account A to account B.

```
T1 :    Read(A);
        A := A-100;
        Write(B);
        Read(B);
        B := B+100;
        Write(B);
```

Let T_2 be a transaction which transfers 10 percent of the balance from account A to account B. It is defined as

```
T2 :    Read(A);
        temp := A * 0.1;
        A := A-temp;
        Write(A);
        Read(B);
        B := B+temp;
        Write(B);
```

Schedule-1 (A Serial Schedule in which T_1 is followed by T_2)

T_1

```
Read(A);
A := A-100;
Write(A);
Read(B);
B := B+100;
Write(B);
```

T_2

```
Read(A);
temp := A * 0.1;
A := A - temp;
Write(A);
Read(B);
B := B + temp;
Write(B);
```

Schedule-2 (A Serial schedule in which T_2 is followed by T_1)

T_1	T_2
	Read(A);
	temp := A * 0.1;
	A := A - temp;
	Write(A);
	Read(B);
	B := B + temp;
	Write(B);
Read(A);	
A := A - 100;	
Write(A);	
Read(B);	
B := B + 100;	
Write(B);	

Schedule-3 (A concurrent schedule equivalent to Schedule-1)

T_1	T_2
Read(A);	
A := A-100;	
Write(A);	
	Read(A);
	temp := A*0.1;
	A := A-temp;
	Write(A);
Read(B);	
B := B+100;	
Write(B);	
	Read(B);
	B := B+temp;
	Write(B);

Schedule-4 (A concurrent schedule)

T_1	T_2
Read(A);	
A := A -100;	
	Read(A);
	temp := A * 0.1;
	A := A - temp;
	Write(A);
	Read(B);
Write(A);	
Read(B);	
B := B+100;	
Write(B);	
	B := B+temp;
	Write(B).

There are two forms of schedule equivalence

1. Conflict Serializability
2. View Serializability.

Conflict Serializability

Conflicting Instructions

The instructions I_i and I_j are said to be conflicting instructions if they are operations by different transactions on the same data item and atleast one of these instructions is a write operation.

Conflict equivalence

If a schedule S can be transformed into a Schedule S', by a series of swaps of non-conflicting instructions, we say that S and S' are conflict equivalent

Conflict Serializable

A schedule S is conflict Serializable if it is conflict equivalent to a serial schedule.

View Serializability

Consider two schedules S and S', where the same set of transactions participates in both schedules. The schedules S and S' are said to be view equivalent if three conditions are met.

1. For each data item Q, if transaction T_i reads the initial value of Q in schedule S, then transaction T_i, must, in schedule S', also read the initial value of Q.

2. For each data item Q, if transaction T_i executes read(Q) in schedule S and if that value was produced by a write(Q) operation executed by transaction T_j, then the read(Q) operation of transaction T_i must, in schedule S', also read the value of Q that was produced by the same write(Q) operation of transaction T_j.

3. For each data item Q, the transaction that performs the final write(Q) operation in schedule S must perform the final write(Q) operation in schedule S'.

View Serializable

A schedule S is view serializable if it is view equivalent to a serial schedule.

Lock-Based Protocols

Lock :- A lock is a variable associated with a data item.

Locks are of two types

1. Shared-mode lock
2. Exclusive-mode lock.

1. Shared-mode lock (S) :- If a transaction T_i has obtained a shared-mode lock on data item Q then T_i can read but cannot write Q

2. Exclusive-mode lock (X) :- If a transaction T_i has obtained an exclusive-mode lock on data item Q, then T_i can both read and write Q.

The concurrency-control manager grants the locks to the transactions based on lock compatibility matrix.

T_i \ T_j	S	X
S	True	False
X	False	False

Lock-compatibility Matrix.

1. If transaction T_i has acquired a shared lock on data item 'ϕ' and if transaction T_j requests for Shared lock on data item 'ϕ' then it can be granted.

2. If transaction T_i has acquired a shared lock on data item 'ϕ' and if transaction T_j requests for exclusive lock on data item 'ϕ', then it cannot be granted.

3. If transaction T_i has acquired an exclusive lock on data item 'ϕ' and if transaction T_j requests for Shared lock on data item 'ϕ', then it cannot be granted.

4. If transaction T_i has acquired an exclusive lock on data item 'ϕ' and if transaction T_j requests for exclusive lock on data item 'ϕ' then it cannot be granted.

```
lock_X(A);
Read(A);
A:=A-50;
Write(A);
Unlock(A);
lock_X(B);
Read(B);
B:=B+50;
Write(B);
Unlock(B);
```

Two-phase locking protocol

This protocol requires that each transaction issue lock and unlock requests in two phases:

1. Growing phase :- A transaction may obtain locks, but may not release any lock.
2. Shrinking phase :- A transaction may release locks, but may not obtain any new locks.

Lock point of the transaction.

The point in the schedule where the transaction has obtained its final lock is called the lock point of the transaction

Two phase locking is of two types

1. Strict two-phase locking
2. Rigorous two-phase locking

Strict two-phase locking

All exclusive mode locks taken by a transaction be held until that transaction commits.

Rigorous two-phase locking

All locks taken by a transaction be held until the transaction commits.

Lock conversion

Lock upgrade

Lock downgrade.

Lock upgrade :- converting shared lock to exclusive lock

Lock downgrade :- converting exclusive lock to shared lock.

```
lock_X(A);
lock_X(B);          } Growing phase
Read(A);
A := A-50;
Write(A);
Read(B);
B := B+50;
Write(B);
Unlock(A);
Unlock(B);          } shrinking phase
```

Graph-Based protocol.

If the information about the order in which the database items will be accessed is known then graph based protocol can be implemented.

Let $D = \{d_1, d_2 \cdots d_n\}$ be the set of all data items

If $d_i \rightarrow d_j$, then any transaction accessing both d_i and d_j must access d_i before accessing d_j.

The partial ordering among the data items may be viewed as directed acyclic graph known as Database graph. The graphs here considered are trees and hence it is known as tree protocol.

Tree protocol

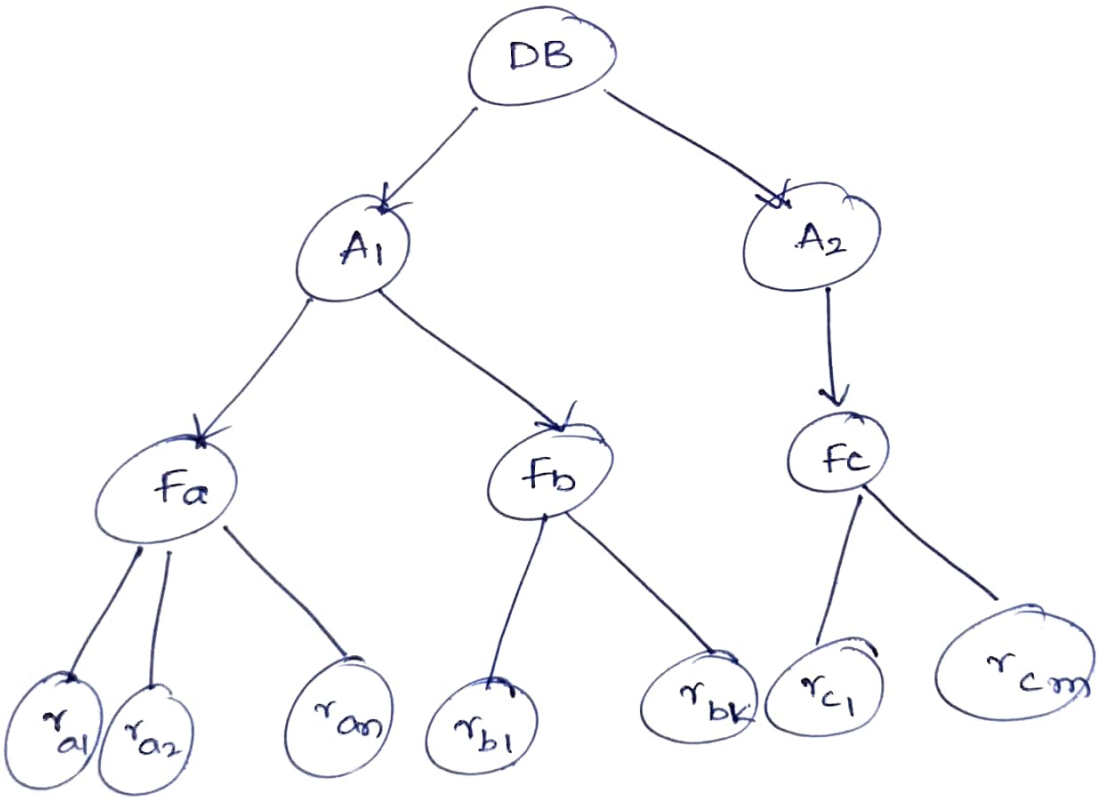
1. The only lock allowed is exclusive lock.
2. The first lock by T_i may be on any data item
3. Subsequently, a data item Q can be locked by T_i if and only if the parent of Q is currently locked by T_i.
4. Data items may be unlocked at any time.
5. A data item that has been locked and unlocked by T_i cannot subsequently be relocked by T_i.

Multiple Granularity

The system is allowed to define multiple levels of granularity.

The highest level represents the entire database.

The next level is area and each area inturn has file and each file has records.



Granularity Hierarchy.

DB — Database

A_1, A_2 — Areas

F_a, F_b, F_c — Files

$r_{a1}, r_{a2}, r_{an}, r_{b1}, r_{bk}, r_{c1}, r_{cm}$ — Records.

Lock-compatibility Matrix.

	IS	IX	S	SIX	X
IS	T	T	T	T	F
IX	T	T	F	F	F
S	T	F	T	F	F
SIX	T	F	F	F	F
X	F	F	F	F	F

IS - Intention Shared

IX - Intention Exclusive

SIX - Shared Intention Exclusive.

Each transaction T_i can lock a node Φ by following these rules:

1. The lock compatibility matrix should be observed.
2. It must lock the root of the tree first and can lock it in any mode.
3. It can lock a node Φ in S or IS mode only if it currently has the parent of Φ locked in either IX or IS mode.
4. It can lock a node Φ in X, SIX or IX mode only if it currently has the parent of Φ locked in either IX or SIX mode.
5. It can lock a node only if it has not previously unlocked any node.
6. It can unlock a node Φ only if it currently has none of the children of Φ locked.

Timestamp - Based Protocol

A protocol which is based on the ordering of the transactions is known as Timestamp-based protocol.

Timestamp :- A unique number assigned to each transaction before the transaction starts its execution is known as timestamp.

The timestamp of a transaction T_i is denoted by $TS(T_i)$.

There are two methods for assigning timestamps.

1. The value of the system clock is assigned as the timestamp of the transaction.
2. The value of the logical counter is assigned as the timestamp of the transaction. The value of the logical counter is incremented after a new timestamp has been assigned.

To implement this scheme, We associate each data item Q, two timestamp values :

* **W_timestamp(Q)** - It denotes the largest timestamp of any transaction that executed Write(Q) successfully.
* **R_timestamp(Q)** - It denotes the largest timestamp of any transaction that executed read(Q) successfully.

Timestamp-ordering protocol ensures that any conflicting read and write operations are executed in timestamp order.

1. Suppose that transaction T_i issues read(Φ)

   (a) If $TS(T_i) < W\_timestamp(\Phi)$, then the read operation is rejected and T_i is rolled back.

   (b) If $TS(T_i) \geq W\_timestamp(\Phi)$, then the read operation is executed and $R\_timestamp(\Phi)$ is set to the maximum of $R\_timestamp(\Phi)$ and $TS(T_i)$

2. Suppose that transaction T_i issues write(Φ)

   (a) If $TS(T_i) < R\_timestamp(\Phi)$, then the write operation is rejected and T_i is rolled back.

   (b) If $TS(T_i) < W\_timestamp(\Phi)$, then the write operation is rejected and T_i is rolled back.

   (c) otherwise, the system executes the write operation and sets $W\_timestamp(\Phi)$ to $TS(T_i)$.

Thoma's Write Rule

The modification to the timestamp-ordering protocol is called Thoma's Write Rule.

Suppose that transaction T_i issues Write(Φ)

1. If $TS(T_i) < R\_timestamp(\Phi)$ then the system rejects Write operation and T_i is rolled back.
2. If $TS(T_i) < W\_timestamp(\Phi)$ then the write operation is rejected.
3. otherwise, the system executes write operation and sets $W\_timestamp(\Phi)$ to $TS(T_i)$.

Phantom Phenomenon

* Insertions can lead to phantom phenomenon, in which an insertion logically conflicts with a query even though the two transactions may access no tuple in common.

* Index-locking technique can solve this problem by requiring locks on certain index buckets.

Validation-Based Protocol

In this scheme, each transaction T_i executes in three different phases in its life time, depending on whether it is a read-only or an update transaction.

The phases are :-

1. **Read phase :-**
   * During this phase, the system executes transaction T_i
   * It reads the values of the various data items and stores them in local variables, without actual updates of the actual database.
   * It performs all write operations on temporary local variables, without updates of the actual database

2. **Validation phase :-**

   Transaction T_i performs a validation test to determine whether it can copy to the database the temporary local variables that hold the results of write operations without causing a violation of serializability.

3. **Write Phase :-**

   If transaction T_i succeeds in validation, then the system applies the actual updates to the database. Otherwise, the system rollsback T_i.

Three different timestamps are associated with every transaction T_i.

(1) Start(T_i) – The time when T_i started its execution

(2) Validation(T_i) – The time when T_i finished its read phase and started its validation phase.

(3) Finish(T_i) – The time when T_i finished its write phase.

The validation test for transaction T_j requires that, for all transactions T_i with $TS(T_i) < TS(T_j)$, one of the following two conditions must hold:

1. $\boxed{Finish(T_i) < Start(T_j)}$

Since T_i completes its execution before T_j started, the serializability order is indeed maintained.

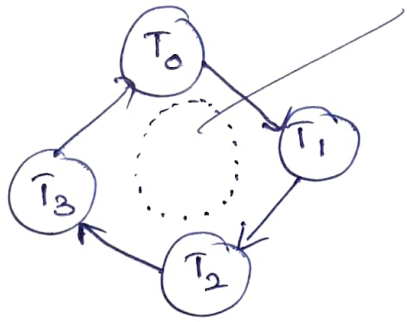
2. $\boxed{Start(T_j) < Finish(T_i) < Validation(T_j)}$

This condition ensures that the writes of T_i and T_j do not overlap. Since the writes of T_i do not affect the read of T_j and since T_j cannot affect the read of T_i, the serializability order is indeed maintained.

Deadlock

A system is in a deadlock state if there exists a set of transactions such that every transaction in the set is waiting for another transaction in the set.

Let $T_0, T_1 \ldots T_n$ be a set of transactions such that T_0 is waiting for a data item that T_1 holds and T_1 is waiting for a data item that T_2 holds and .., and T_{n-1} is waiting for a data item that T_n holds and T_n is waiting for a data item that T_0 holds. None of the transactions can make progress and this situation is called Deadlock.



Deadlock Detection :-

Deadlock can be described in terms of a directed graph called a Wait-for graph.

$$G = (V, E)$$

V = The set of transactions $\{T_1, T_2 \ldots T_n\}$

E = The set of Edges $\{(T_i \rightarrow T_j)\ (T_j \rightarrow T_K) \ldots\}$

An edge $T_i \rightarrow T_j$ exists if transaction T_i waits for a data item being hold by transaction T_j

A deadlock exists in the system if and only if the wait-for graph contains a cycle.

Deadlock Recovery

The most common solution for deadlock is to rollback one or more transactions to break the deadlock.

Three actions need to be taken

1. Selection of a victim.
2. Rollback
3. Starvation.

1. Selection of a victim

Given a set of deadlocked transactions, we must determine which transaction to rollback to break the deadlock.

Many factors may determine the cost of a rollback including

(a) How long the transaction has computed and how much longer the transaction will compute before it completes its designated task.

(b) How many data items the transaction has used

(c) How many more data items the transaction needs for it to complete.

(d) How many transactions will be involved in the rollback.

2. Rollback :- Once we have decided that a particular transaction must be rolled back, we must determine how far this transaction should be rolled back. There are two options for rollback

(a) Total Rollback

(b) partial Rollback.

Total Rollback :- The entire transaction is rolled back and then restarted.

Partial Rollback :- The transaction must be rolled back to the point to break the deadlock.

It requires the system to maintain additional information about the state of all the running transactions.

The system should decide which locks the selected transaction needs to release in order to break the deadlock.

3. Starvation :- In a system where the selection of victims is based primarily on cost factors.

It may happen that the same transaction is always picked as a victim. As a result the transaction never completesand it is known as starvation.

The most common solution is to include the number of rollbacks in the cost factor.

Deadlock Prevention

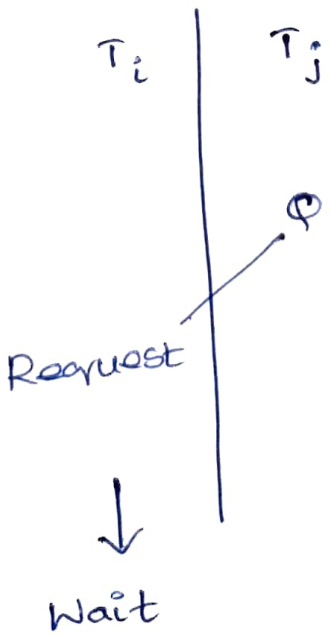
It includes mechanisms that ensure that the system will never enter a deadlock state.

Two different deadlock prevention schemes using timestamps have been proposed.

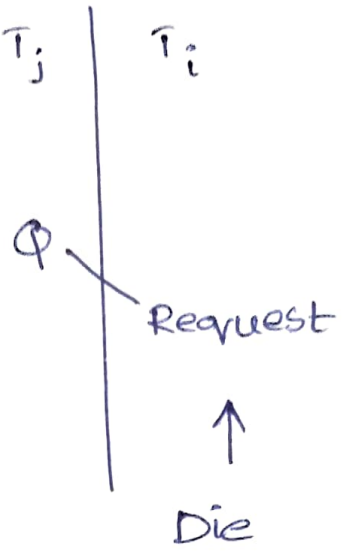
1. Wait-die scheme
2. Wound-Wait scheme.

Wait-die scheme :-

* It is a Nonpreemptive technique.
* When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if T_i has a timestamp smaller than that of T_j. Otherwise T_i is rolled back.



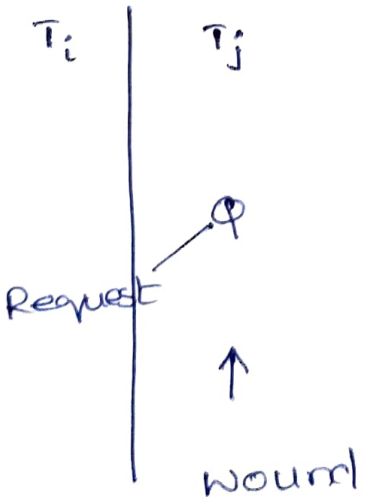
$TS(T_i) < TS(T_j)$



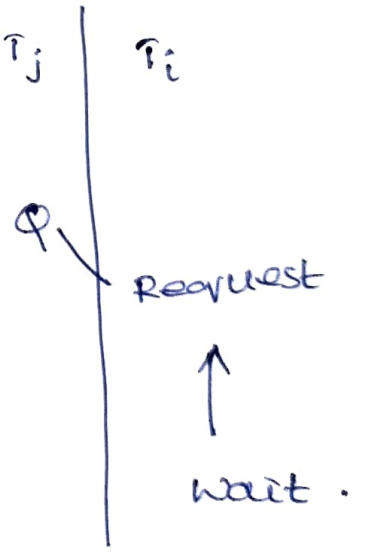
$TS(T_i) > TS(T_j)$

(2) Wound-Wait Scheme :-

* It is a preemptive technique.
* When transaction T_i requests a data item currently held by T_j, T_i is allowed to wait only if it has a timestamp larger than that of T_j. otherwise T_j is rolled back.



$TS(T_i) < TS(T_j)$



$TS(T_i) > TS(T_j)$

Failure :-

Whenever the system fails to function according to its specification and does not deliver the expected service, that situation is called the failure of the system.

Types of Failures

Failures can be classified into four types. They are :

1. Hardware failures
2. Software failures
3. System crash
4. Transaction failures

Hardware Failures

They include memory errors, disk problems, crashes bad sectors etc.

Software Failures

They include operating system failures, application program failures etc.

System Crash

A system failure can occur due to hardware/software or power failure.

Transaction Failures

If a transaction fails due to various reasons like hardware failure, software failure etc then it is known as transaction failure.

Storage types

Storage media can be classified into three types.

1. Volatile storage
2. Non Volatile storage
3. Stable storage.

Volatile Storage

Information residing in volatile storage does not usually survive System crashes.

E.g. Main memory and cache memory.

Non volatile storage

Information residing in non volatile storage survives system crashes.

E.g. Disk, Magnetic tapes.

Stable storage

Information residing in stable storage is never lost

E.g. RAID

RAID Stands for Redundant Array of Inexpensive DisKS

Data Access

The database system resides permanently on non volatile storage and is partitioned into fixed-length storage units called blocks.

Physical blocks

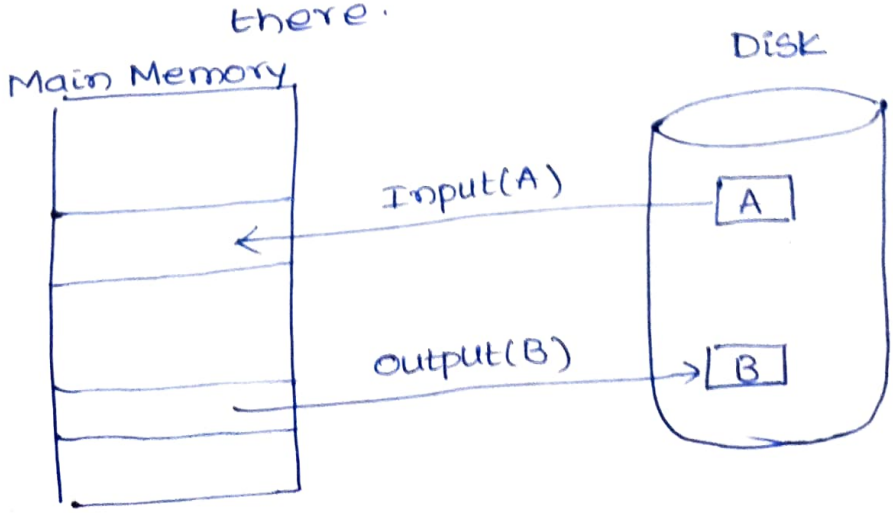
The blocks residing on the disk are referred to as physical blocks.

Buffer blocks

The blocks residing temporarily in main memory are referred to as buffer blocks.

Block movements between disk and main memory are initiated through the following two operations.

1. Input(B) :- It transfers the physical block B to main memory
2. Output(B) :- It transfers the buffer block B to the disk and replaces the appropriate physical block there.



Database Recovery Management

It helps in recovering the database from failures.

Log-Based Recovery

Log:- A log is a sequence of log records, recording all the update activities in the database.

There are several types of log records.

1. $\langle T_i \text{ start} \rangle$
   Transaction T_i has started

2. $\langle T_i \text{ commit} \rangle$
   Transaction T_i has committed.

3. $\langle T_i \text{ abort} \rangle$
   Transaction T_i has aborted.

4. $\langle T_i, X_j, V_1, V_2 \rangle$
   Transaction T_i has performed write operation on data item X_j.
   V_1 is the value of data item before write
   V_2 is the value of data item after write.

This is known as update log record.

< Transaction identifier, Data-item identifier, old value, New value >.

Log-based Recovery operates in two modes.

1. Deferred Database Modification.
2. Immediate Database Modification.

Deferred Database Modification

It records all database modifications in the log, but deferring the execution of all write operations of a transaction until the transaction partially commits.

It uses the operation Redo(T_i)

Redo(T_i) – It sets the value of all data items updated by transaction T_i to the new values.

Immediate Database Modification.

It records all database modifications in the log and all writes are output to the database while the transaction is in active state.

It uses two operations

(1) Undo(T_i) – Setting the old values of data items

(2) Redo(T_i) – Setting the new values of data items.

* Transaction T_i needs to be undone if the log contains the record $<T_i$ start> but does not contain the record $<T_i$ commit>

* Transaction T_i needs to be redone if the log contains both the records $<T_i$ start> and $<T_i$ commit>

Checkpoint:-

A checkpoint declares a point before which the DBMS was in consistent state and all the transactions were committed.

It is a procedure of compressing the transaction log file by transferring the old transactions to permanent storage.

Fuzzy checkpoint :-

The checkpointing technique requires that all updates to the database be temporarily suspended while the checkpoint is in progress.

If the number of pages in the buffer is huge, a checkpoint may take a long time to finish, which can result in an unacceptable interruption in processing of transactions.

The checkpointing technique can be modified to permit updates to start once the checkpoint record has been written, but before the modified buffer blocks are written to disk. The checkpoint thus generated is known as fuzzy checkpoint.

ARIES recovery algorithm

A – Algorithm for
R – Recovery and
I – Isolation
E – Exploiting
S – Semantics.

ARIES algorithm uses

1. It uses a log sequence number (LSN) to identify log records and the use of LSNs in database pages to identify which operations have been applied to a database page.
2. It supports physiological redo operations, which are physical in that the affected page is physically identified, but can be logical within the page.
3. It uses a dirty page table to minimize unnecessary redos during recovery. Dirty pages are those that have been updated in memory and the disk version is not up-to-date.

4. It uses a fuzzy-check pointing scheme that records only information about dirty pages and associated information and does not even require writing of dirty pages to disk.

The recovery consists of three phases:

1. Analysis phase :-

It determines the earliest log record from which the next pass must start.

2. Redo phase :-

Starting at the earliest LSN, the log is read forward and each update redone.

3. Undo phase :-

The log is scanned backward and updates corresponding to loser transactions are undone.