23/10/20.

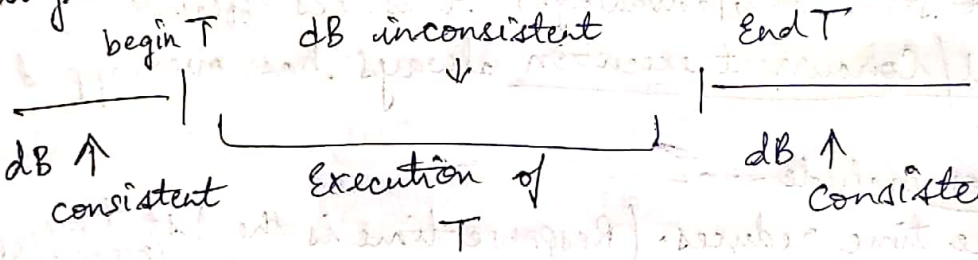
Transaction Management (Unit-4)

* Transaction is a sequence of operations/instructions.
* Transactions are atomic. It is either complete or null process. If interrupted, it rolls back.
* Transaction is a single logical unit of work which access database & possibly modify the database.

ACID Properties :-

↳ Atomicity, Consistency, Isolation, Durability.

* Transaction will modify the database so as to it make it from one consistent state to another consistent state.



In between dB can be inconsistent.

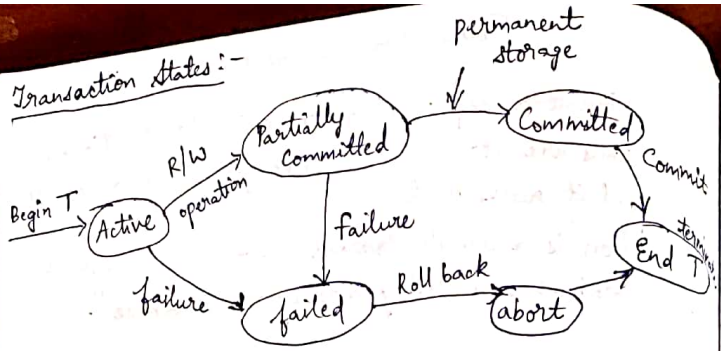
Atomicity: It states transaction has to be executed completely or not at all. No partial execution. If there is partial execution then roll back.
Transaction Management Component (TMC) ensures the atomicity.

Consistency: Transaction working on a dB has to lead from one consistent state to another consistent state.
It is the duty of Programmer/Developer to ensure consistency.

Isolation: To implement the logical isolation, the intermediate results of the individual transactions should not be made available to each other which run concurrently.

Durability: If the transaction is committed, then it has to persist forever, it can't roll back even if the system fails.
In this case, we need to write a compensating transaction, since rolling back is not possible. This is durability.

Transaction states :-



Concurrency :-

Serial execution of transaction means there is no overlapped execution and hence will always lead to consistent state (only advantage). It is less effective.

Non-Serial / Concurrent execution always has overlapped execution.

Advantages include ---

① Response time reduces. (Response time is the 1st time (resource/CPU) when we get response).

② Average waiting time reduces. (Waiting time while waiting for resources/CPU)

③ Resource utilisation increases.

④ Efficiency (η) increases.

"Executions overlapped with other transactions is known as Concurrency."

Problems associated with concurrency ---

* It may lead to inconsistency.

Hence we allow controlled concurrency following ACID properties :-

① Dirty read problem / Reading uncommitted data.

solution → do not read uncommitted data.
→ if you read then do not commit.

② Unrepeatable Read problem.

↳ Occurs because of isolation break.

solution → do not read repeatedly (or roll back)

③ Phantom read problem.

↳ trying to read something which does not exist.

④ Lost update problem.

↳ trying to access lost updates.

Schedule :-

Dictates the order in which things happen.

→ Serial schedule
→ Non-Serial / Concurrent schedule.

* In S.S, if there are 'n' transactions, then $n!$ or $\lfloor n$ serial transaction schedules are possible.
* Consider $T_1, T_2, \dots T_n$ transactions containing $n_1, n_2 \dots n_n$ operations. Then the possible non-serial schedules are $\frac{\lfloor n_1+n_2+\dots+n_n}{\lfloor n_1 \cdot \lfloor n_2 \cdot \dots \lfloor n_n} - \lfloor n$

Q1) Given NSS, is it serializable?
Q2) How to generate a NSS which is serializable?

Serializability :- (flexible).

All the NSS which are output equivalent to one of Serial schedule then we say it is a serializable schedule. This is serializability.

Serializable schedule
- Conflict S.S. (CSS) — stricter
- View S.S. (VSS) — strict

* CSS and VSS will lead to consistency but it does not mean that the if the schedule is not CSS or not VSS, it will lead to inconsistency.

Conflict Serializability (CSS) :-

Read — R, Write — W.

RR — Non-Conflicting
RW, WR, WW — conflicting.

R(A), W(A) — Conflicting
R(A), W(B) — Non-Conflicting because A & B are diff.

If through series of swapping of Non-Conflicting operations (NCOs), if we are able to convert NSS into one of the possible S.S, then the schedule is CS.

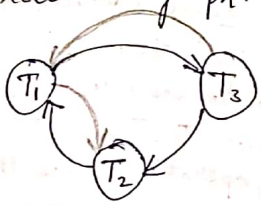
Q) When NSS is CS ?

Ans- A NSS is CS if it is conflict equivalent to any possible serial schedule.

ex:

T_1	T_2	T_3
R(x)		
		R(z)
		W(z)
	R(y)	
R(y)		
	W(y)	
		W(x)
	W(z)	
W(x)		

Precedence graph.

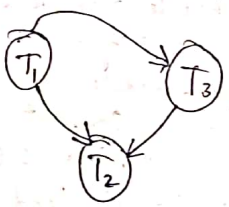


	is CS ?
cyclic	X
Acyclic	✓

ex:

T_1	T_2	T_3
R(x)		
	R(y)	
		R(y)
	W(y)	
W(x)		
		W(x)
	R(x)	
	W(x)	

Precedance graph.



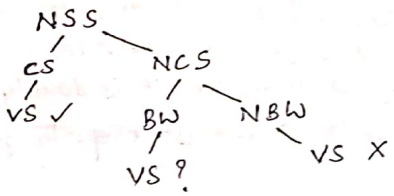
It is acyclic and hence CS.

This also tells the order T_1, T_3, T_2.

i.e we are applying Topological sort to know the order.

* If a schedule is CS then it is VS also but the vice-versa is not true.

→ S is not CS
- and S do not have a blind write (VS X)
- but have a blind write (may or may not be VS)



View Serializability :-

NSS is VS iff it is view equivalent to one of the possible serial schedule.

Let S be a NSS and S' be a SS.

	S	S'
Initial Read (IR)	T_1	T_1
Final Write (FW)	T_2	T_2
Update (U) (or) Intermediate read	$T_1 \rightarrow T_2$	$T_1 \rightarrow T_2$

S S'
T_1T_2 T_1T_2

* Blind write is writing directly / blindly without initial reading.

ex:

S		
T_1	T_2	T_3
R(a)		
	W(a)	
W(a)		
		W(a)

	A
IR	T_1
FW	T_3
U	α

T_1 T_2 T_3 ✓ VS

1 2 3 => T_1 T_2 T_3
1 3 2 => T_1 T_3 T_2
2 1 3
2 3 1
3 1 2
3 2 1

Recoverability :-

* Irrecoverable schedules lead to inconsistency.
* Solution : Always read committed data.
  Avoid dirty read.
  If there is dirty read, then the order of commit must match the order of dirty read.
* If dirty read is not there, then it is directly recoverable.
* Hence Recoverability is a must to property for a schedule.

Cascadelessness :-

Suppose $T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4 \rightarrow T_5$

T_1 failed ← cascading Rollbacks → leads to wastage of time.

(i.e. T_2 reading from T_1, T_3 from T_2 .... So if T_1 fails then all have to rollback. This is cascading rollback. This reduces the efficiency).

Cascading Rollback also known as Cascading Abort.

* If there is dirty read
  - C.R ✓
  - Recoverable & Non-Recoverable.
* To avoid C.R, always read committed data.
* All cascadelessness schedules are recoverable but all recoverable schedules are not cascadelessness schedule.

Strict Schedule :-

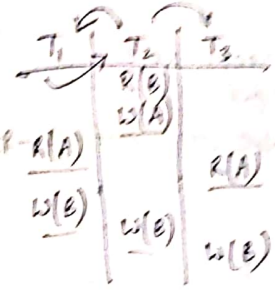
Strict Shedule doesn't allows dirty read and dirty write operations.

Solution : Read and write only committed data.
Otherwise lost update problem occurs.

* All serial schedules are strict but all strict schedules are not serial.

Eg: ① $S : R_2(B)\ \ W_2(A)\ \ R_1(A)$
$R_3(A)\ \ W_1(B)\ \ W_2(B)\ \ W_3(B)$.
Is it view serializable ?

Sol^n :



$T_1 \leftrightarrow T_2$ is cyclic
∴ hence not CS.

	A	B
IR	✗	T_2
FW	T_2	T_3
I/M R	$T_2 \rightarrow T_1$ ✗ $T_2 \rightarrow T_3$	$T_2 \rightarrow T_3$

Hence it is VS. ⟸ ∴ $T_2 \rightarrow T_1 \rightarrow T_3$

②

T_1	T_2	T_3
R(A)		
	R(A)	R(B)
W(A)		
	R(C)	
	R(B)	
	W(B)	
		W(C)

	A	B	C
IR	T_1	T_3	T_2
FW	T_1	T_2	T_3
I/M R	✗	✗	✗

∴ Not view serializable.

③ $S : R_1(x), R_2(x), R_1(z), R_3(x), R_3(y), W_1(x), W_3(y), R_2(y), W_2(z), W_2(y), C_1, C_2, C_3$;

T_1	T_2	T_3
R(x)		
	R(x)	
R(z)		
		R(x)
		R(y)
W(x)		
		W(y)
	R(y)	
	W(z)	
	W(y)	
C_1	C_2	C_3

dirty read

So it is,
Not Recoverable
C.R.
Not a strict schedule.

(4) PS1: $R_1(x), W_1(x), R_2(x), R_1(y), R_2(y), W_2(x), W_1(y), C_1, C_2$.

T_1	T_2
R(x)	
W(x)	R(x) D.R.
R(y)	R(y)
	W(x)
W(y)	
C_1	
	C_2

It is,
recoverable
C.R
Not strict schedule.

Q1. Is it view serializable? IR (means reading from database)

T_1	T_2	T_3
R(A)		
	R(A)	R(B)
W(A)		
1	R(C)	
	R(B)	
	W(B)	
	2	W(C)

	A	B	C
IR	T_1 T_2	T_3 T_2	T_2
FW	T_1	T_2	T_3
I/M R	$T_2 \to T_1$	$T_3 \to T_2$	α
	1	2	

$T_3 \to T_2$, $T_2 \to T_3$

↳ Contradicting

Hence not V.S.

Q2.

T_1	T_2	T_3	T_4
	R(A)		
R(A)			
W(C)			
		R(C)	
W(B)			
			R(B)
		W(A)	R(C)
	W(D)		
	R(B)		
			W(A)
			W(B)

	A	B	C	D
IR	T_1 T_2	X	X	X
FW	T_4	T_4	T_1	T_2
I/M R	X	$T_1 \to T_4$ $T_2 \to T_4$	$T_1 \to T_4$ $T_1 \to T_3$	X
		$T_1 \to T_4$ $T_2 \to T_4$		

$\therefore T_1 \to T_2 \to T_3 \to T_4$

Hence it is V.S.

Q3.

T_1	T_2	T_3	T_4	T_5
R(A)				
		R(D)		
W(B)				
	R(B)			
		W(B)	R(B)	
	W(C)			
				R(C)
			W(E)	
				R(E)
				R(A)

	A	B	C	D	E
IR	T_1 T_5	X	X	T_3	X
FW	X	T_3	T_2	X	T_4
I/M R	X	$T_1 \to T_2$ $T_3 \to T_4$ $T_4 \to T_3$	$T_2 \to T_5$	X	$T_4 \to T_5$

$\therefore T_1 \to T_2 \to T_3 \to T_4 \to T_5$

All agree on the same order.

Hence V.S. ✓

Q4.

T_1	T_2	T_3	T_4
	R(A)		
R(A)			
W(C)		R(C)	
W(B)			R(B)
		W(A)	R(C)
	W(D)		
	R(B)		
			W(A)
			W(B)

	A	B	C	D
IR	T_1 T_2	X	X	X
FW	T_4	T_4	T_1	T_2
I/M R	X	$T_1 \to T_4$ $T_4 \to T_2$	$T_1 \to T_3$ $T_1 \to T_4$	X

$\therefore T_1 \to T_2 \to T_3 \to T_4$ ✓

and $T_1 \to T_2 \to T_4 \to T_3$ ✓

and $T_1 \to T_3 \to T_2 \to T_4$.

Hence it is V.S.

Time stamping Protocol:-

It says ① Follow ordering between transactions.
② Order must be decided before transaction come into the system.
③ If conflict occurs, resolve using the order.

* For time stamping, we are giving system time which is unique always. (time stamp)

2 TS — ① Transactions
② Data item (Q) — RTS(Q) (Read), WTS(Q) (Write).

* $T_i \to$ Request Read
  - if $T.S(T_i) < W.T.S(Q)$ Reject (then trans. rollback)
  - if $T.S(T_i) \geq W.T.S(Q)$, allow.

ex: $TS(T_1) = 5$, $TS(T_2) = 10$

T_1	T_2
	W(Q)
R(Q)	

$WTS(Q) = 10$

$RTS(T_1) = 5$ ✗ Not allowed. [∵ 5 comes before 10 can't read after 10]

So reject.

But

T_1	T_2
R(Q)	
	W(Q)

$RTS(T_1) = 5$, $WTS(Q) = 10$ } Allowed.

and

T_1	T_2
	W(Q)
	R(Q)

$WTS(Q) = 10$, $RTS(T_2) = 10$ } Allowed.

and

T_1	T_2
W(Q)	
	R(Q)

$WTS(Q) = 5$

$TS(T_2) = 10 > 5$. So allowed.

* $RTS(Q) = Max(RTS(Q)$ or $TS(T_i))$ ← RTS update

ex:

T_1	T_2
W(A)	
	R(A)
R(A)	

$RTS(A) = 10$

$WTS(A) = 5$

$TS(T_1) \geq WTS(A)$

$5 \geq 5$

But $RTS(A) = 10$ not 5

* $T_i \to$ Requested for write
  - if $T.S(T_i) < RTS(Q)$ Reject
  - if $T.S(T_i) < WTS(Q)$ Reject.
  - if $T.S(T_i) \geq RTS(Q) / WTS(Q)$ allow.
  - [$WTS(Q) = T.S(T_i)$] → WTS update.

Initially RTS & WTS values are null or 0.

ex: $T.S(T_1) = 5$, $TS(T_2) = 10$.

T_1	T_2
	R(Q)
W(Q)	

$RTS(Q) = 10$ ~~WTS(Q)~~

$T.S(T_1) = 5 < 10$ reject

and

T_1	T_2
	W(Q)
W(Q)	

$RTS(Q) =$ Null or 0.

$WTS(Q) = 10$.

$T.S(T_1) = 5 < 10$ reject.

and

T_1	T_2
W(Q)	
	W(Q)

$RTS(Q) =$ Null or 0.

$WTS(Q) = 5$.

$T.S(T_2) = 10 > 5$ allow.

T.S Protocol Properties:-

1. Schedule generated by T.S. Protocol is C.S and V.S.
2. " " " " " may or may not be recoverable. (In general we say its not recoverable since we are not sure).
3. It is deadlock free because we either allow or reject but there is no wait, so no deadlock.
4. Performance is not good because of rollback leads to wastage of time.
5. All T.S. schedules are C.S.S. but all C.S.S. are not T.S.S.

Modified T.S Protocol (or) Thomas Write Rule:-

* It is the only protocol which generates V.S.S.

ex:

T_1 (10)	T_2 (20)
R(A)	
	B.W W(A)
W(A)	

→ If it is T.S then this W(A) is rejected but T.W.R. says, don't reject it but just avoid or ignore it because that W(A) is obsolete (no longer useful).

but ex: T_1: R(A), W(A); T_2: R(A), W(A)

→ Here W(A) can't be ignored it has to rollback. because T_2 is not reading from dirty base but from T_1.

* Hence schedules generated by T.M.R are not C.S but V.S ✓.

T.M.R- Properties :-

1. C.S ✗
2. V.S ✓
3. Recoverable ✗
4. Deadlock ✗ (because we are asking to ignore but not to wait).
5. Starvation ✓.
6. Performance is slightly greater than T.S protocol.

Thomas T.M.R states, "if there is any obsolete write before the blind write then we ignore the obsolete write and consider only the blind write".

Locking Protocols :-

Concurrency Control Manager (CCM) grantes to open/lock.

Simple Lock
- Shared Lock (SL) – reading purpose.
- Exclusive lock (EL) – writing purpose

	SL	EL
SL	✓	✗
EL	✗	✗

S(A) – SL on A
X(A) – EL on A
UX(A) – Unlock EL on A
} representation

Properties of simple Locking Protocol :-

1. C.S ✗ (No guarantee of C.S).
2. V.S ✗
3. Recoverable ✗ (May or may not be)
4. Cascadelessness ✗
5. Deadlock ✓
6. Starvation ✓

1) 2PL (two phase Locking) :-

Basic 2PL
- → growing phase (lock can be granted but not released)
- → shrinking phase (lock cannot be granted but be released).

* The moment a transaction takes the last lock (or) the moment it switches from growing phase to shrinking phase, there will be lock point.

Properties :-
1. C.S ✓
2. V.S ✓
3. Recoverable ✗ (May or may not be).
4. Deadlock ✓
5. Starvation ✓
6. Cascadelessness ✗

* Order of lock point is equal to the order of serializability.
* Lock Conversion.

2) Conservative 2 PL (or) Static 2PL :-

It says "take all the locks before performing any operation".

* Advantage is that there will be no deadlock when we are able to start. There can be starvation.
* Hence Conservative 2 PL solves the problem of deadlock. i.e C 2PL is free from deadlock because deadlock always occurs in execution & here execution doesn't start if the locks are not provided first & hence if there's no start – no deadlock.
* All 2 PL are C.S but all C.S.S cannot be generated by 2PL. (2 PL ⊂ C.S)
* Problem of Recoverability & cascadelessness.

3) Rigorous 2 PL :-

It says "cannot release lock until we decide to commit".

* Hence this solves the problem of recoverability and cascadelessness. It is also C.S ✓ and V.S ✓.
* But it is not deadlock free.
* It reduces the order of concurrency. This is the major problem.

4) Strict 2 PL :-

It says "if you have a exclusive lock, you can release it only after committing but if there is a shared lock, then

you can release it anytime".

* C.S ✓
V.S ✓
Recoverability and cascadelessness ✓
Deadlock ✗
It reduces order of concurrency but it is better than rigorous 2PL.

II) Graph-Based Protocol :-

* Once we know the order of how transactions access the data items, then we can pose "Partial order".

Ex:- $D_i \rightarrow D_j$ ⇒ Partial order between D_i & D_j

"If a transaction wants to access both D_i and D_j, then it has to access D_i first and then only D_j" ← condition.

(* DAG is a partial order between data items.
↳ (Directed Acyclic Graph).

a) Tree-Based Protocol :

→ Conditions for each transaction T_i to lock data item Q :

① If 1st time any transaction request for a lock on any data item, then grant it. (provided the data item is free)
② If it is not 1st time, the subsequent request of lock is granted only if the parent is locked.

→ This protocol works only on exclusive lock.
→ Lock can be released any time.
→ Re-locking is not permitted (hence it is C.S).

Properties :-

① C.S ✓ and V.S ✓
② Recoverability (May or may not be).
③ Deadlock ✗ (it is deadlock free).
④ Starvation ✓.
⑤ Order of concurrency (or degree of concurrency) is poor but far better than strict 2PL and rigorous 2PL.

* All TBP schedules are C.S but all C.S.S. cannot be generated by TBP.

Deadlock Handling :-

Deadlock occurs due to ① Mutual Exclusion ② Hold and Wait ③ Circular Wait ④ No premision.
All these 4 conditions together are necessary for deadlock to occur.

* In deadlock the complete system does not work but in starvation only one transaction might not work.

Mutual Exclusion — It says no two transactions can work on the same copy at the same time.

Hold and Wait — ex:

T_1	
A	(Holds A)
B ←	(Needs B)

Circular Wait — ex:

T_1	T_2	
A	B	(Holding)
B ←	A ←	(Waiting value).

No preemption.? — No premision — Cannot forcefully take resource.

Disadvantages of Deadlock :-

* In real-time system, system cannot tolerate deadlock because life and money is on stack — like ex: in detecting seismic waves (tsunamis, earthquakes etc.).
* Even in not real-time systems, resource utilisation is reduced & system efficiency reduces.

* *

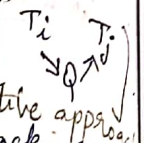
Deadlock avoidance / prevention (Pessimistic Approach).
— Here we are trying to prevent/avoid because we know deadlock is going to happen. So thinking negatively. Hence is the name. We use this approach when we know that deadlock is frequent. Cost of deadlock is high. (real-time systems).

Deadlock detection and recovery (Optimistic Approach).
— Here we are ready to recover if deadlock occurs. So +ve. Cost of deadlock is low. Life & money are not on stack, only resource utilisation is less.

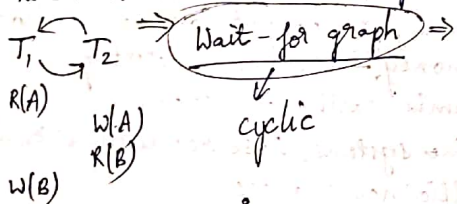
Solutions :- (for preventive approach).

* To avoid Hold & Wait — take all resources before the starting the execution.
* To avoid Circular Wait — Impose ordering on data item and transaction lock data items in the sequence consistent with ordering. ex: Tree-based protocol.
* No method to avoid Mutual Exclusion.
* * • Wait and die (non-preemptive approach)
if $T.S(T_i) > T.S(T_j)$ then T_i roll back.
if $T.S(T_i) < T.S(T_j)$ then T_i wait

– If rollback then T.S will be same to avoid starvation and hence releases all resources. So no deadlock.
• Wound and wait (preemptive approach).
if $T.S(T_i) > T.S(T_j)$ then T_i wait
if $T.S(T_i) < T.S(T_j)$ then T_j roll back.



• Lock timeout (preemptive approach) – preventive approach – lock within time limit otherwise roll back. If the lock is not granted within the time, the transaction is said to timeout & it itself rollbacks & restarts.

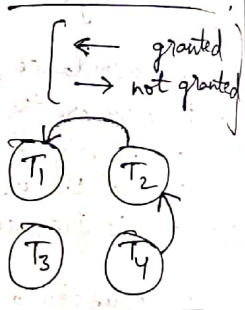


⇒ Simple way to detect deadlock.

ex:

Transaction	DataItems	Lock Mode
T_1 ←	Q	S.L
T_2 ←	P	EL
→	Q	EL
T_3 ←	Q	S.L
T_4 →	P	E.L

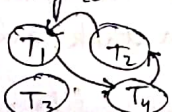
Deadlock detection



So one of the solution is $T_3 \to T_1 \to T_2 \to T_4$.
and no deadlock because no cycle.

but in same ex: if we add

T_4 ←	R	EL
T_1 →	R	EL



So here there is deadlock.

Solutions :- (for recovery)
Steps:
→ Select victim → minimum cost.
→ Roll back → Partial Roll back (rollbacks only till save point).
→ Full Roll back (abort)
→ Starvation. (Should ~~not~~ select victim such that one transaction must not starve again & again).

Validation Based Protocol (Concurrency Control Protocol) :-
* It is a optimistic technique of concurrency control, because ~~this~~ the assumption here is "very rare conflicting operations will be there, most of the operations are read only".
This protocol works on 3 phases :
① Read Phase — • Here we read dataitems from databases, and store/create local copy. (Hence recoverable always).
• All writes will be on local copy (Both read & write can be done in this phase).
② Validate Phase — • Test for serializability.
• If pass, write to database is permitted otherwise discard the update and restart.
③ Write / Write Back / Update / Finish Phase — Apply all updates done by a transaction to the actual database.

The two scenarios which give serializability are :
1) Finish (T1) < Start (T2) — even if 1 condition is satisfied it is serializable.
2) Start (T2) < Finish (T1) and Finish (T1) < Validate (T_2).
* This test is on individual transactions.
* Not suitable for longer transactions.
* Very rare rollbacks (or) very less conflicting operations.

Left Over topics

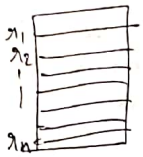
Unit-I.
Functional Component of DBMS :-
├ Storage Manager → Manage the storage space.
└ Query Processor

Unit -5

29/12/20

File Structure & Indexing:-

* File is a data structure to store data.
ex: In a file system having 'n' random records,



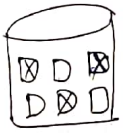
Then for searching a random record
Best time $O(1)$
Avg. time $O(n)$ $[\because O(\frac{n+1}{2}) = O(n)]$
Worst time $O(n)$

If the file is sorted then, Best time ~~$O(n)$~~ $O(1)$
Worst time $\log_2(n)$

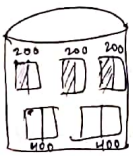
File Allocation

- Contiguous allocation
- Non-contiguous allocation (linked allocation)
- Index allocation.

In Continuous allocation,

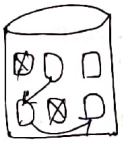


If we want to allocate 3 blocks, even if 3 blocks are free, we can't allocate because there are not contiguous.
This is called external-fragmentation.



Here we 200, 200, 400 available to allocate for a 1200 B storage but we can't do it since it is not contiguous.
This is ~~is~~ called internal fragmentation.

In Non-contiguous allocation,



This also suffers from internal fragmentation due to links but is free from external fragmentation.

In Index allocation,

each file has a special block and is free from internal fragmentation too.

File → Sorted
File → Unsorted

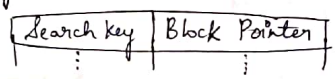
Sorted	Unsorted
* Searching will be fast. * Insertion and deletion is not simple. * File can be sorted based on only one attribute and w.r.t other attributes, file remains unsorted.	* Searching is not fast. * Insertion and deletion will be easy.

Spanned Mapping	Unspanned Mapping
* Records are stored accross the blocks if they do not fit in one block. * We have to access more than one block to access a record. * We are saving the space.	* Records are not stored accross the blocks if they do not fit in one block. * No need to access more than one block to access a record. * Not saving space but saves time.

Indexing:-

* Indexing is used for faster access.
* Indexing is optional.
* For a file, indexing can be done on any number of attributes.
* Index file is always sorted.
* Index file size must always less than Main file.

Index file structure is



Sparse Indexing:-

* Main file is sorted to apply sparse indexing.
* We do not include each and every search key in the index file and hence reduces size.
* If main file is sorted we may go to sparse indexing but it is not compulsory.

Dense indexing	Sparse Indexing.
When every search key get a entry in index file then it will be dense indexing. (main file is unsorted).	When every search key do not get a entry in index file then it is sparse indexing. (main file is sorted).

Types of indexing:-

- Sparse Vs dense indexing.
- Single Vs Multilevel indexing
- Primary Vs Secondary Vs cluster indexing.

* Multilevel indexing — Creating index for a index file.
* Single level indexing — has only 1 index for a main file.

Primary Indexing:-

* The main file is sorted based on primary key.
* The primary key is used as search key, thats why the name is primary indexing.
* No. of entries in the index file = No. of blocks acquired by main file.
* Access time to access a random record if the index file need n blocks = $\text{ceil}(\log_2(n)) + 1$.
* It is the example of sparse indexing.

Cluster Indexing:-

* The ~~main~~ DB file is sorted based on non-key attribute.
* The search key ~~may or may~~ is not ~~be~~ key attribute.
* No. of entries in the index file = no. of unique search keys (when main file is unsorted or has repeated keys.)
  = no. of blocks acquired by main file (when main file is sorted). i.e when all keys are unique)
* It is the example of both sparse indexing and dense indexing.
* Block access time $\geq \text{ceil}(\log_2(n)) + 1$.

Secondary indexing:-

* The file is unsorted.
* The search key may or may not be key attribute.
* It is example of dense indexing.
* No of entries in the index file = no. of blocks acquired by main file.
* Block access time $\geq \lceil \log_2 n \rceil + 1$.

ex:

All these indexing are ordered indexing i.e search key follows a order.

Hashing:-

Hash index: search key is in order according to the hash function.

O(1) {independent of n} where n is size of data.

* In DBMS, there is no collision, there will only be bucket overflow.

ex: 120, 130, 140, 62, 51

$h(K) = K \% 2$

0	120	130	140	62
1	51			

if bucket size = 4 no overflow but if it <4 then overflow but no collision.

Static Hashing (Solutions for overflow):-

1. Chaining (open Hashing)
2. Open addressing (Closed Hashing)
   - Probing is the solution for overflow
   - A) Linear Probing
     $h(K, i) = ((h(K) + c * i)) \bmod m$.
   - B) Quadratic Probing.
     $h(K, i) = ((h(K) + c * i^2)) \bmod m$.
   - C) Double hashing
     $(h(K) + h'(K) * i) \bmod m$.
3. Dynamic Hashing $\Rightarrow g(K) = \{ h_0(K), h_1(K), h_2(K) \ldots \}$.

Types of Hashing :-

* Direct Hashing
* Modulo division hashing
* Mid square hashing — Key distribution of key is non-uniform.

B-Tree and B+Tree (Tree based Indexing) :-

* They are extension to binary search tree.
* ~~B-Tree is also called~~ For M-way search tree.
  If the order is M then, atmost M-1 (search) keys and M children.
* We need B and B+Tree because they allow automatic multilevel indexing.

B-Tree :-

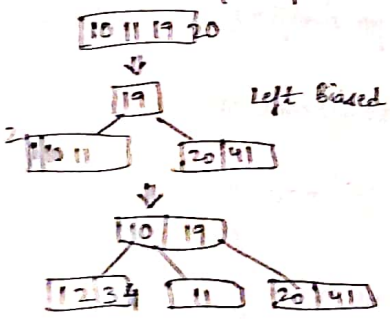
Restrictions / Conditions :

1. It has to be balanced tree i.e all leaf node must be at same level.
2. Each node, except root must be filled with atleast half i.e $\lceil M/2 \rceil$
3. Root >= 1 keys.
4. All nodes may contain atmost M-1 keys & m children.
5. Keys inside the node are sorted.
6. We create tree using bottom up approach.
   → min. no. of keys = ceil(M/2) − 1.

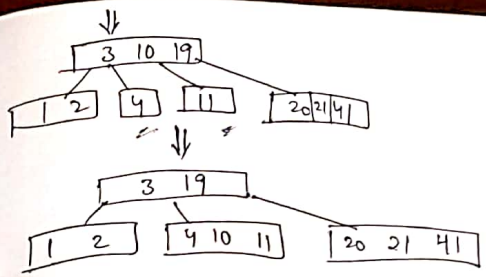
Ex: Create a B-Tree using 10, 11, 19, 20, 41, 1, 2, 3, 4, 21 and order 4.

Sol: order 4 => 4 max children, max 3 keys at each node

$\lceil M/2 \rceil = 2$.

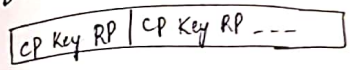


Left Biased B Tree (LBBT)
(Left having more elements).



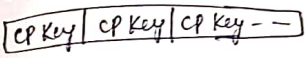
[∵ each node $\lceil M/2 \rceil$]

B-Tree Structure

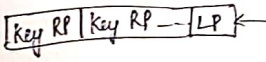


for both Non-leaf and leaf node.

B+Tree Structure

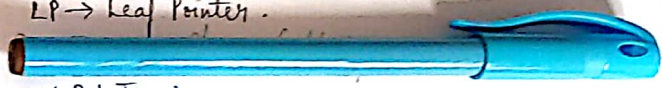


Non-leaf



leaf.

where CP → Child Pointer, RP → Record Pointer
LP → Leaf Pointer.



Advantages of B+ Tree :-

* Height of tree is reduced.
* Faster access to search queries.

(Recovery --- Notes / PPT in classroom)