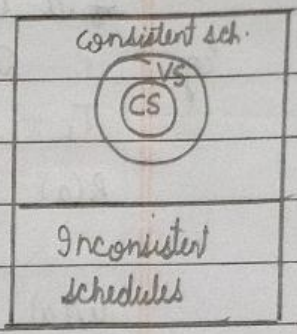
View Serializability

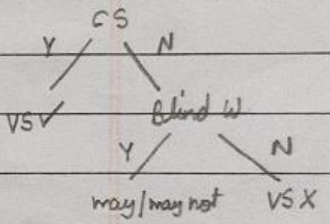
A schedule is a view serializable schedule if it is view equivalent to any of the serial schedules.



Conditions for view equivalence:

- Initial reads (for all variables involved) must be the same for both schedules
- Final write (for all variables involved) must be the same for both schedules
- Updated read / intermediate read:
  If in schedule S1, T_1 is reading value of any variable updated by write operation of T_2 then, in S_2 the order should be same i.e. T_1 must read value of variable after the write operation of T_2.

- If a schedule is CS, it is VS
- If a schedule is not CS, and doesn't have a blind write, it is not VS. (CS ✗ schedule must have [atleast] a blind write [not the only condition to be VS if not VS ✗])



Example:

S

T_1	T_2
IR R(a)	
W(a) ←	R(a)
	W(a) FW
IR R(b)	
W(b) ←	R(b)
	W(b) FW

S'

T_1	T_2
IR R(a)	
W(a) ←	
IR R(b)	
W(b)	R(a)
	W(a) FW
	R(b)
	W(b) FW

Two transactions ∴ 2! serial schedules are possible

S must be view eq to any of the 2 serial sch.

IR ✓
FW ✓
Int. R ✓

S is view equivalent to S' (serial)
∴ S is VS

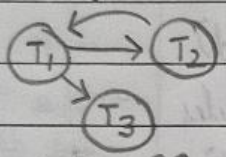
In prev eg it was CS so VS as well

If It is better to check for CS first and then VS.

Eg:

S

T_1	T_2	T_3
IR R(a)		
	W(a) Blind	
W(a)		
		W(a) Blind FW



not CS

Contains blind write

∴ may / may not be VS

Now check ~~for~~ each serial schedule one by one until u get view equivalence

S'

T_1	T_2	T_3
IR R(a)		
W(a)		
	W(a)	
		$W(a)_{FW}$

IR ✓

FW ✓

Int R ✓ (as ~~Int R~~ is not Int)

S is v. eq. to S'

S is VS

dirty Read
N / Y
RV — May / May not
if order of commit same RV
No RX

Recoverable Schedule

- A schedule which can handle system failures.
- Schedule in which transactions commit only after all transactions whose changes they read commit.
- If some transaction T_j is reading value updated or written by some other transaction T_i, then the commit of T_j must occur after the commit of T_i.

A schedule must be recoverable

S

T_1	T_2	
10 R(A)		A=10
20 A=A+10		
20 W(A) ←		dirty read
	R(A)	20
	A=A-5	15
	W(A)	15
	C	
Failure		
T_1 rolls back but T_2 can't C		A=15

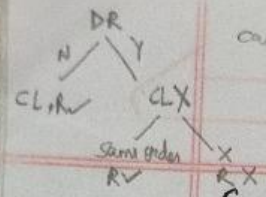
If T_1 commits before T_2 → Recov ✓

T_2 is dependent on T_1

It is mandatory property

condition : must not contain dirty reads

If dirty reads are present → order of rollbacks must be same as order of dirty reads (cascaded rollbacks)



cascading rollback one after other

Cascadeless Schedule
(optional property)

condition : schedule must not have dirty reads

removes cascading rollbacks (occurs due to dirty read)

S

T_1	T_2	T_3
R(A)		
W(A)		
	R(A) DR	
	W(A)	
failure		R(A) DR
✗	C	
		C

$T_1 \rightarrow T_2 \rightarrow T_3$

Recoverable ✓

cascadeless ✗

Cascading Rollback Schedule :

If in a schedule, failure of one transaction causes several other dependent transactions to rollback or abort, such schedule is called as cascading schedule.

- It leads to wastage of CPU time.
- It occurs due to dirty reads

T_1	T_2	T_3
R(A)		
W(A)		
C		
	R(A) DRX	
	W(A)	
	C	R(A) DR X
		C

dirty read ✗

Recoverable ✓

Cascadeless ✓

but concurrency lost

Cascadeless schedule :

It is a schedule in which dirty reads are not present and hence cascading rollbacks do not occur.

Commit & read

- Dirty reads are eliminated by committing the changes made by a transaction before the other transaction reads the changed value.

- It is an optional property (it reduces concurrency)
- It allows only committed reads. However, it allows uncommitted writes.
- A cascadeless schedule is always recoverable, vice versa not true.

T_1 T_2
R(a)
W(a)
C
W(a)
R(a)
C

Strict Schedule

Even if a schedule is cascadeless, it may contain lost updates (due to blind writes)

	T_1	T_2	
a=10		W(a)	a=100
a=10	R(a) a=a+5		
a=15	W(a)		
a=100	C		
		R(a)	a=100
		C	

Here when T_1 commits, it wants to store a=15 but due to blind write of T_2 a=100 gets stored

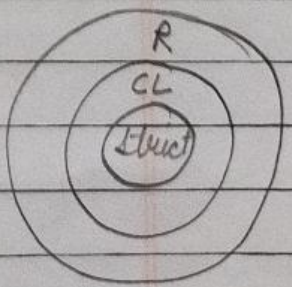
This schedule is cascadeless (no dirty reads)

A strict schedule is a schedule in which a write operation is always followed by a commit i.e. another transaction performs operations on a data value only after changes made by the earlier transaction has been committed.

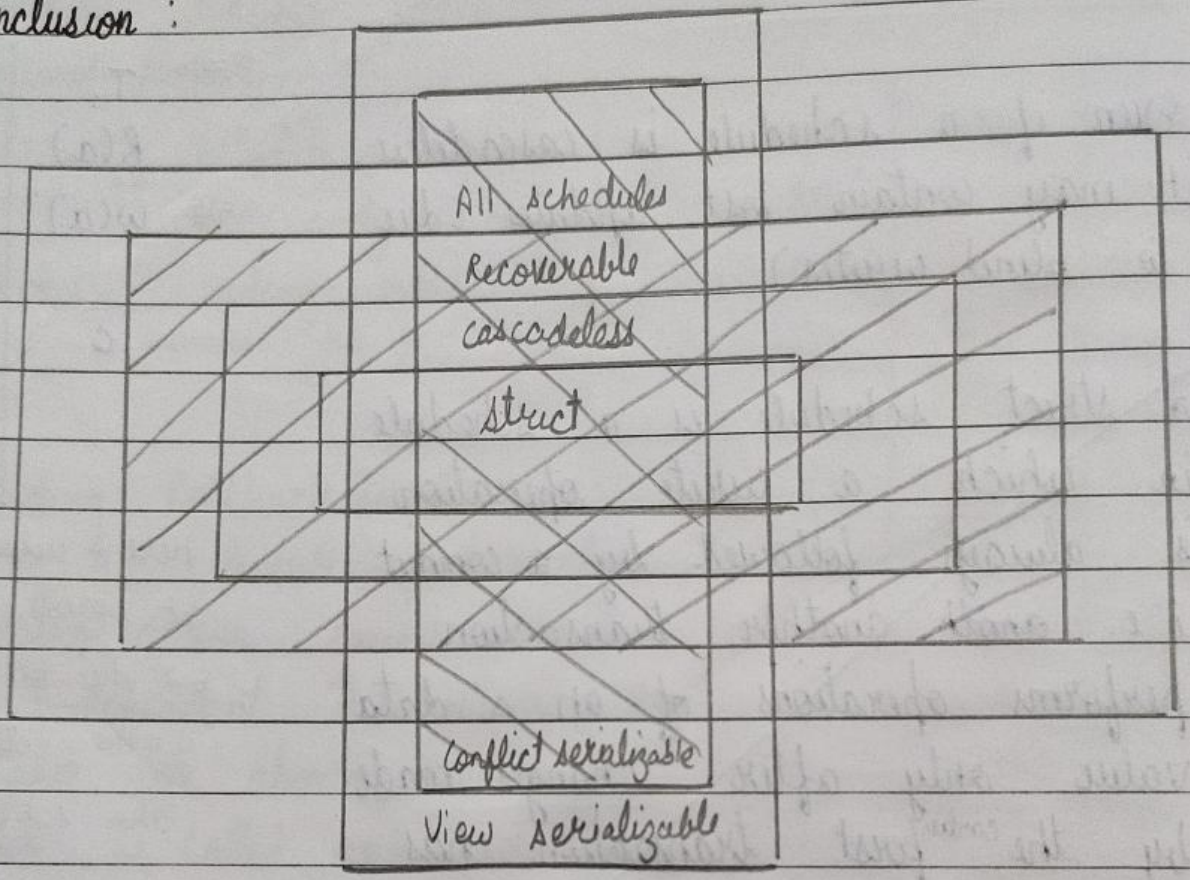
T_1	T_2
R(a)	
	R(b)
W(a)	
	W(b)
C	
	R(a)
	C

A strict schedule is a schedule in which changes made by a write operation of a transaction is committed before it is read/write by another transaction.

- To avoid lost updates and dirty reads.



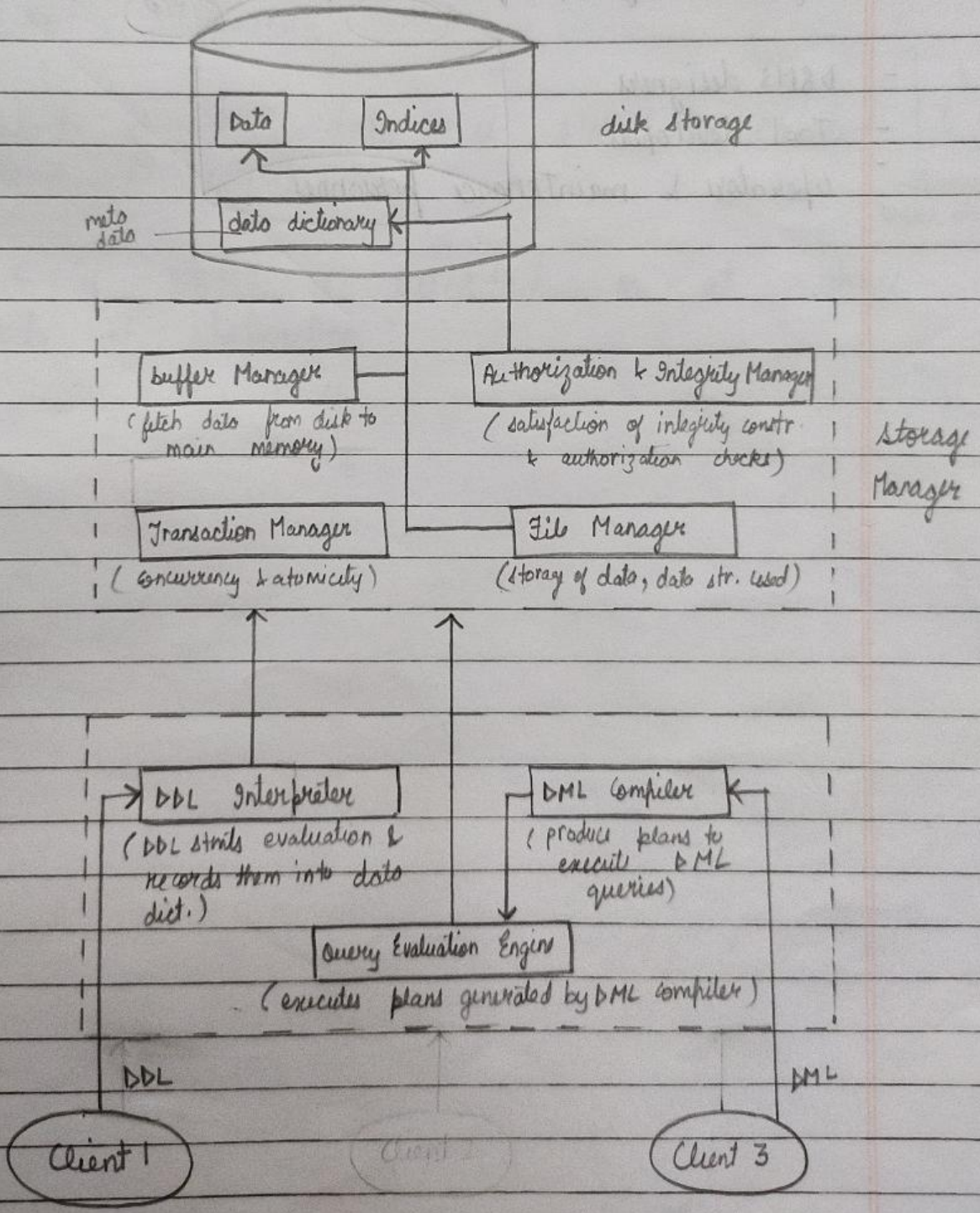
Conclusion :


All schedules
Recoverable
cascadeless
Struct
Conflict serializable
View serializable

Database System Structure

DBMS takes queries, process them & produce plans (sequence of operations) to implement them, fetches the data from database and sends to appl. prog.

sql (non procedural) —DML compiler→ sequence of operations to execute the query (procedural)



People concerned with DBMS

- DBA (Database Administrator) - team
  manages the resources
  - primary resource - database
  - sec. —— - DBMS
- Database designers - usually from DBA team
- End users - querying job
- system analysts / software developers - design app program

behind the scene not concerned with the content

- DBMS designers
- Tool developers
- Operators & maintenance personnel

3 Explain Data abstraction

Data abstraction :
Hiding the complexities of the system from the client & providing an abstract view of only relevant information only is known as data abstraction.

Details of how data is stored, created & maintained is hidden.

Data is DBMS is described at three levels of abstraction :

Name : Payal
Roll no : 23609

ASSIGNMENT : 1 UNIT : 1

1 Difference between DDL & DML

DDL	DML
• (Data Definition Language)	• (Data Manipulation Language)
• It is used to create, delete & modify database schema	• It is used to insert, updat, delet & retrieve data from database.
• It defines the column (attributes) of the table.	• It add or updat the row (tuple) of the table.
• It doesn't have any further classification	• It is further classified as procedural & non-procedural DML
• Commands present is DDL: Create Drop Rename Alter etc.	• Commands present in DML: Updat Insert Delete Merge etc.
• It doesn't use where clause.	• It uses where clause
• Changes are reflected in the schema	• Changes are reflected in the instance.

Value set / domain of attribute

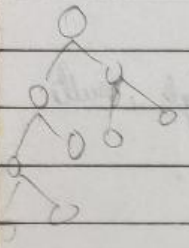
Each attribute has its value set V.

If attribute → A (simple attribute)
entity set → E
Value set → V
A : E → P(V) → Power set of V
it covers all combos of values - null, single, multi

If attribute A is composite
attribute → A (composite)
entity set → E
value sets for its components → $V_1, V_2, V_3 \dots V_n$
Value set → $V = P(V_1) \times P(V_2) \times P(V_3) \times \dots \times P(V_n)$
$A : E \to P(V) = P[P(V_1) \times P(V_2) \times P(V_3) \times \dots \times P(V_n)]$

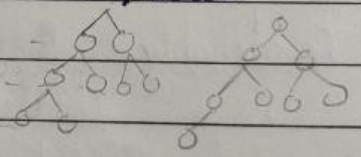
B
KB 2^{10}
MB 2^{20}
GB 2^{30}
TB 2^{40}
PB 2^{50}

designing
- ER Diagram
- Conversion of ER → relational
- Basics of rel. model & FD
- Keys / types
- Normalisation
- Lossless decomposition & dependency preserving
- Indexing & physical str.
- SQL, RA, RC
- Transaction
- Concurrency Control



OLAP (Online Analytical Processing)	OLTP (Online Transaction Processing)
(data warehouse, legacy data)	(database)
for policies etc, Research	- historical data	- current data
	- subject oriented	- application oriented.
Data Mining	- decision making	- day to day operations.
	- size TB, PB	- MB, GB
	- CEO, Managing Director, General Manager - access	- clerk, managers.
	- only read operations	- read / write

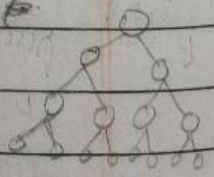
Complete binary tree - all full except last level - as left as possible



for h ht: max nodes = $2^{h+1}-1$; min nodes: 1, 2, 4, 8 = 2^h

for n nodes: min ht = $\lceil \log_2(n+1) \rceil - 1$; max ht = $\log_2 n$

Full / Proper / strict - each node has 2 nodes except leaf (either 0 or 2 children)



no. of leaf nodes = no. of internal nodes + 1

for ht h: max nodes = $2^{h+1}-1$; min nodes: 1, 3, 5, 7 = $2h+1$

for n nodes: max ht $(n-1)/2$; min ht $\lceil \log_2(n+1) \rceil - 1$

Perfect: all - 2 childs, leaf at same level — CBT ✓ FBT ✓ (vice versa)

Degenerate: all has 1 child — left skewed BT, right skewed

NORMALISATION

Process of removing redundant data from the relations to improve storage efficiency, data integrity, and scalability and speed.
It usually involves decomposition of relations.

First Normal Form

→ A table is said to be in first normal form when it follows following conditions :

- Values of all the attributes must be atomic
- Every column must have unique name
- Every record of a relation must be unique
- All values of a particular column must be from same domain
- Order of rows & columns is irrelevant
- A relation must have a primary key

→ When an ER model is mapped into relational model, it is present in 1NF.

→ 1NF can be attained by

- Creating multiple columns for multi-valued attributes and composite attributes
- Creating multiple tables (for multivalued attributes)

Examples:

Roll	Name	Contact
101	A	9562338912, 9123457891
102	B	9143287321, 8215331448

not in 1NF
Contact is multivalued

↓

Roll	Name	Contact1	Contact 2
101	A	9562338912	9123457891
102	B	9143287321	8215331448

SId	Name	Hobby
1	X	Singing, Painting, Reading
2	Y	Dancing, Writing

↓

SId	Name
1	X
2	Y

S_No	Hobby	SId
101	Singing	1
102	Painting	1
103	Reading	1
104	Dancing	2
105	Writing	2

Roll	Name	Course
1	A	CN/OS
2	B	DBMS/DS

↓

Roll	Name	Course
1	A	CN
2	A	OS
3	B	DBMS
4	B	DS

Second Normal Form

→ A relation is said to be in 2NF if.

- It is in 1NF
- It doesn't contain partial dependencies

→ Partial dependency : A functional dependency is said to be a partial dependency if a part of candidate key determines the non prime attribute.

→ Problem with partial dependency

A	B	X
1	Null	a
Null	2	b
3	Null	c
4	5	d

Here AB combined is a composite CK & can determine X. But part of AB (i.e. 'A' or 'B') can not determine X because they contain 'Null' values.

→ To find whether a table is in 2NF
(when it is given that table is in 1NF)
- Find all the CKs
- Find prime and non prime attributes
- Non prime attributes must not be involved in partial dependencies.

Example : R(ABC) A B → C

- Candidate key : $(AB)^+ = ABC$

∴ AB is a CK

A and B are essential attributes of CK ∴ it is the only CK of given relation.

- Prime attributes : A, B
  Non prime attributes : C

  Since, in FD $B \to C$, B is a part of CK and C is a non prime attribute, it is a partial dependency.
  ∴ given relation is not in 2NF.

→ To convert in 2NF

$R(ABC) \Rightarrow R_1(BC)$ & $R_2(AB)$
$B \to C$ $\quad\quad$ $B \to C$

A	B	C
a	1	x
b	2	y
a	3	z
c	3	z
d	3	z
e	3	z

R

A	B
a	1
b	2
a	3
c	3
d	3
e	3

R_1

B	C
1	x
2	y
3	z

R_2

Here, R contains redundant data in the records corresponding to B = 3
After decomposition, redundancy is reduced and 2NF is attained.

Third Normal Form

A relation is said to be in 3NF if,

- It is in 2NF
- It does not contain transitive dependencies

Transitive dependency: A functional dependency is said to be transitive if a non prime attribute determines another non prime attribute

Problem with transitive dependency

A	B	C
a	1	x
b	1	x
c	1	x
d	2	y
e	2	y

Here A is the CK & B, C are non prime attributes where $B \rightarrow C$
It leads to redundancy of data.

To find whether a relation is in 3NF
(when already present in 2NF)

- Find all the CKs
- Find prime and non prime attributes
- Check whether any non prime attribute is being determined by another non prime attribute

or

Check for every dependency from $\alpha \rightarrow \beta$

- either α is a super key
- or β is a prime attribute

example:

R(ABC) $A \to B$ $B \to C$

Candidate key : $A^+ = ABC$

Since A is an essential attribute ∴ A is the only CK

Prime attributes : A

Non Prime attributes : B, C

In FD $B \to C$ B which is a non prime attribute determines C which is another non prime attribute. It is a transitive dependency Hence given relation is not in 3NF

To convert into 3NF

R(ABC) → R_1(BC), R_2(AB)

$A \to B$
$B \to C$

R_1: $B \to C$

R_2: $A \to B$

R

A	B	C
a	1	x
b	1	x
c	1	x
d	2	y
e	2	y
f	3	z
g	3	z

R_1

A	B
a	1
b	1
c	1
d	2
e	2
f	3
g	3

R_2

B	C
1	x
2	y
3	z

BCNF (Boyce Codd Normal Form)

→ A relation is said to be in BCNF if

- It is in 3NF
- Prime attributes are not determined by any non prime attribute or part of CK.

→ To determine whether a relation is in BCNF (when already present in 3NF)

- Find all the CKs
- Find prime & non prime attributes.
- Check whether any prime attribute is being determined by non prime or part of CK.

Example: R(ABC) $AB \rightarrow C$ $C \rightarrow B$

Candidate keys: $(AB)^+ = ABC$
$(AC)^+ = ABC$
∴ AB & AC both are CKs.
Prime attributes: A, B, C
Non prime attributes: —

Here in FD $C \rightarrow B$, C is a part of CK and determines a prime attribute B
∴ It is not in BCNF

Or In FD $\alpha \rightarrow \beta$
α should be super key.

Authenti Autho access control DAC MAC RBAC Models
intrusion detection SQL injection

DATABASE SECURITY

Database security refers to the range of tools, controls and measures designed to establish and preserve database confidentiality, integrity and availability.

It handles various issues such as
- legal and ethical issues
- Policy issues
- System - related issues
- The need to identify multiple security levels

Threats to databases :

Loss of integrity : Database integrity refers to the requirement that information be protected from improper modification.
Use of corrupted data could result in inaccuracy, fraud or erroneous decisions.

Loss of availability : Database availability refers to making objects available to a human user or a program to which they have a legitimate right.

Loss of confidentiality : Database confidentiality refers to the protection of data from unauthorised disclosure.

Control Measures:

- Access control
- Inference control
- Flow control
- Data encryption

ACCESS CONTROL

Access control is a method of allowing access to sensitive data of database only to those people who are allowed to access the data and to restrict the access to unauthorised persons.

Authentication : It is a method of verifying the identity of a person who is accessing the database.

- It usually requires user's login details.
- It verifies whether the user is authenticated or not.
- eg: log in id, password

It can be

→ single factor authentication
(only user id and password is required)

→ 2 factor authentication
(more info like OTP etc is required)

Authorisation : It is a method of verifying the authorities of a user for accessing the resources.

- It requires user's privilege or security levels.
- It determines the permissions user have.

Access control
- Whenever a user needs to access a database system, he has to apply for a user account
- DBA creates the user account and password if there is legitimate need to access the data
- The user must login with the account number and password which is validated by the DBMS.

audit trail:
It is a record which contains all the updates applied to the database and particular users who applied each update.
It is useful for database audit or in case of system failures

Access control Mechanisms:

Discretionary Access Control (DAC)

It is a type of security access control based on granting and revoking of privileges.

Types:

1) Account level: At this level, the DBA specifies the particular privileges that each account holds independent of relations in the database

- It may include CREATE SCHEMA, CREATE TABLE, CREATE VIEW, DROP, MODIFY privilege etc

2) Relation level: At this level, DBA can control the privilege to access each individual relation or view in the database.

- It may include SELECT on R, Modification on R etc

- Grant: It is used to provide the permission to use a privilege.
- Revoke: It is used to take back the permission granted.

Propagation of privilege

A privilege on R can be given with or without grant option. If a privilege is provided with a grant option, the privilege can be propagated.

But if the privilege is revoked later, it must be taken back from all the accounts to which it was propagated.

Mandatory Access Control (MAC)

- It is a hierarchical model. The hierarchy is based on security levels.
- All users are assigned a security or clearance level and all objects are assigned a security label.
- Generally four security classification levels are used:
  - Top Secret (TS)
  - Secret (S)
  - Confidential (C)
  - Unclassified (U)

$$TS \geq S \geq C \geq U$$

Simple security property

- A subject S is not allowed to read access to an object O unless
  Class (S) $\geq$ Class (O)
- This implies that no subject can read an object whose security classification is higher than the subject's security clearance.

Star property

- A subject S is not allowed to write an object O unless
  class (S) $\leq$ Class (O)
- It prohibits a subject from writing an object at a lower security classification than the

subject's security clearance.
- Violation to this rule would allow information to flow from higher to lower classifications

MAC ensures a high degree of protection
∴ they are suitable for military and high security types of applications

Role-Based Access Control (RBAC)

- Role based access control restricts the access based on a person's role within an organization rather than individual users.
- Access can be based on several factors such as authority, responsibility, job competency etc.
- Various roles are created and security privileges are granted to the role name and any individual assigned to this role would automatically have those privileges granted.
- Users may also be assigned to multiple groups roles.
- Separation of duties can be implemented by providing mutually exclusive roles.
- Mutually exclusive roles can be:
  → authorisation time exclusion (static)
  roles that are mutually exclusive can't be part of a user's authorisation at the same time under authorisation time exclusion.
  → runtime exclusion (dynamic)
  roles can be authorized to one user but can't be activated at same time.

Name : Payal
Class : CSE 2 (Sem. III)
Roll No : 23609

UNIT : 1

Ques: Explain E R diagram. Define various types of attributes and relationships.

UNIT : 2

Ques: Define Normalisation. Explain various normal forms.

UNIT : 3

Ques: Define Transaction. Explain ACID properties Also explain different states of a transaction.

UNIT : 4

Ques: Explain Database security? Briefly explain DAC, MAC & RBAC models.

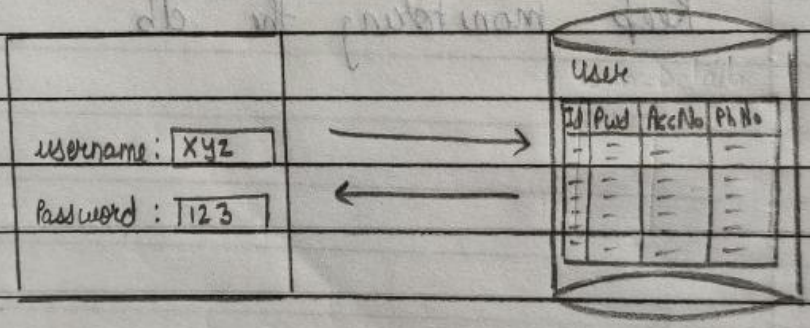
SQL Injection

SQL Injection is a type of attack on database in which user's data is exploited by hacker through web page inputs by injecting malicious SQL commands into it.

→ Web programs and applications that access a database can send commands and data to databases as well as display data retrieved from the database through the web browser.

→ In SQL injection, a string input is injected through the application which manipulates the SQL statement to attacker's advantage.

Example:



Now, if the attacker enters:

username : " OR 1 = 1 --

" to close username string

-- to comment out rest of the query

A condition which is always true is injected with the OR operator so that the result is always true.

earlier query: select * from users where Id = "xyz" and ~~pass~~ pwd = "123"

later query: select * from users where Id = "xyz" or 1=1-- and pwd = "123"

1=1 always true ∴ attacker will login successfully even if Id & pwd are incorrect

Risks :

- Database finger printing
- Denial of service
- Bypass authentication
- Identifying injectable parameters (gathers info abt back end)
- Executing remote commands
- Performing privilege Escalation

Prevention

- Bind variables
- tools & softwares → security event manager
  Netsparker Vulnerability scanner
- limit the size of fields
- Keep monitoring the db

Intrusion Detection

An Intrusion Detection System (IDS) is a system that monitors network traffic for suspicious activity and issues alerts when such activity is discovered. It is a software application that scans a network or a system for harmful activity or policy breaching.

Any malicious venture

B TREE (Balanced Tree)

- It is a self balancing search tree.
- It is used to maintain multi level indices for the data stored in secondary memory.
- It reduces the disk access time.
- Size of a node is kept equal to block size of disk.
- Height of B-tree is low. (It is broader)

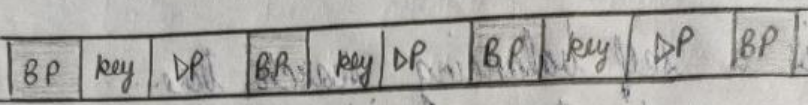
Properties :

- All leaf nodes are at same level.
- It is defined by order 'm' which depends on the block size of disk.
- Maintains data in sorted order.
- If order of b tree = m
  - min children : leaf = 0
    root = 2
    internal nodes = $\lceil \frac{m}{2} \rceil$
  - max children = m
  - min keys : root = 1
    all other $\lceil \frac{m}{2} \rceil - 1$
  - max keys = m-1
- Time complexity for search, insert, delete is $O(\log n)$ n = nodes

Each node contains search key as well as record pointer & link to child nodes.

structure of node

m=4



BP = Block Pointer / Tree pointer points to the child node

key = data value used for searching

DP = Data pointer / Record pointer points to the record in where actual data is stored.

Ques: Consider a B-Tree with key size = 10 bytes, block size 512 bytes, data pointer is of size 8 bytes and block pointer is 5 bytes. Find the order of B-Tree

node size $\leq$ block size

size of block = 512

node size = n (size of BP) + (n-1) [size of key + size of DP]

$= n(5) + (n-1)[10 + 8]$

$= 5n + 18n - 18 = 23n - 18$

Here n = order of b-tree

$\therefore 512 \geq 23n - 18$

$n \leq 530/23$

$n \leq 23.04$

$\therefore$ n must be 23

Only leaf node contains record pointer.

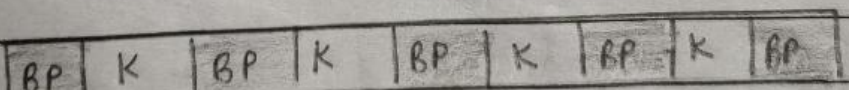
B^+ TREE

- In B tree ~~data~~ is stored in leaf node as ~~data~~ record / data pointers are stored along with the keys in leaf nodes as well as internal nodes. This reduces the no. of entries of a node ∴ increases height.
- But in B^+ trees data pointers are stored only in leaf node.
- Leaf nodes contain all the key values along with their corresponding record pointers.
- So leaf nodes form the first level index and internal nodes form other multi levels.
- Height remains balanced & less as compared to B Tree
- All leaf nodes are linked.

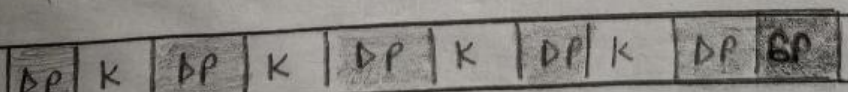
Structure of nodes :

Internal node :

m = 5



Leaf node :



Search access time is lesser

CONCURRENCY CONTROL TECHNIQUE

The concurrency control protocols ensure the atomicity, consistency, isolation durability & serializability of concurrent schedules.

concurrency ✓ properties ✓ time ↓ logic ✓

- Time stamp protocol
- Lock based protocol
  - 2PL (
  - graph based
- validation protocol

TIME STAMP PROTOCOL

- Time stamp ordering protocol orders the transactions in the increasing order of their creation i.e. based on their time stamps.
- The priority of older transaction is higher that's why it executes first in case of conflicts
- System clock is used for timestamps bcz it is unique & never repeats itself.

→ Two kinds of time stamps are maintained:

1) Time stamp with transaction
   - For each transaction T_i, a timestamp is associated denoted by $TS(T_i)$ based on the system clock.
   - This timestamp remains constant throughout the execution of that transaction.
   - If T_j enters after T_i then,
     $TS(T_i) < TS(T_j)$
   - It helps in determining the order of serializability.

2) Time stamp with data item:

For each data item Q, 2 timestamp are maintained.

- W-timestamp (Q): The largest timestamp of any transaction that executed write(Q)
- R-timestamp (Q): The largest timestamp of any transaction that executed read(Q).

Rules

If T_i request for read(Q) Two cases:

1) if $TS(T_i) < WTS(Q)$

means T_i wants to read a value already overwritten by a transaction having lesser priority

In this case request must be rejected & T_i must rollback.

eg: $TS(T_i) = 5$

$WTS(Q) = 10$

$TS(T_i)=5$	$TS(T_x)=10$
T_i	T_x
	$W(Q)$
$R(Q)$	

not allowed

In case of conflicts T with higher priority i.e. lesser TS must perform the operation

2) if $TS(T_i) \geq WTS(Q)$ if equal → same transaction

means T_i wants to read a value already overwritten by a transaction having higher priority

In this case operation is allowed & RTS(Q) will be updated to max (TS(T_i), RTS(Q))

eg: $TS(T_i) = 10$

$WTS(Q) = 5$

$TS(T_x)=5$	$TS(T_i)=10$
T_x	T_i
$W(Q)$	
	$R(Q)$

allowed

If T_i request for Write(Q): 4 cases

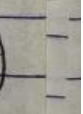
1) if $TS(T_i) < RTS(Q)$
means value of Q is already read by a transaction with lower priority.
In this case transaction is rejected & roll back.

2) if $TS(T_i) < WTS(Q)$
means value of Q is already overwritten by a transaction having lesser priority.
In this case transaction is rejected & rollback.

3) if $TS(T_i) \geq RTS(Q)$ or 4) $TS(T_i) \geq WTS(Q)$
operation is allowed & $WTS(Q)$ must be updated to $max(TS(T_i), WTS(Q))$

Properties

- It ensures conflict serializability.
- It ensures view serializability
- Possibility of dirty read, no restriction on commit
  ∴ irrecoverable and cascading rollback schedules are possible.
- Either an operation is allowed or rejected, so deadlock doesn't occur.
- Every Timestamp protocol-schedule is CS but all CS schedules may not be generated by TS.
- May lead to starvation & relatively slow.



THOMAS WRITE RULE

- It modifies time-stamping protocol to make some improvements & may generate schedules which are VS but not CS & provide better concurrency
- Modifies time-stamping protocol in obsolete write case:
  Ti requests for write (Q)
  if $TS(T_i) < WTS(Q)$
- In this case rather than rollback of transaction, write operation is ignored & transaction is proceeded.

Q=10

T_1	T_2
$TS(T_1)=5$	$TS(T_2)=10$
Q=20 W(Q)	
	W(Q) Q=30

~~W(Q)~~ Allowed order

Q=10

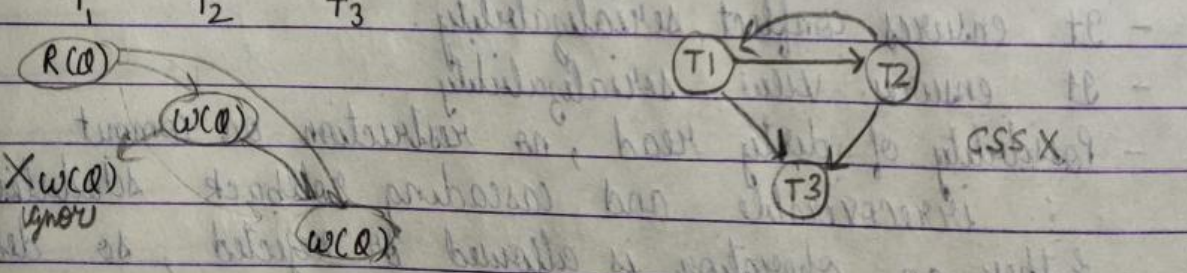
T_1	T_2
$TS(T_1)=5$	$TS(T_2)=10$
	W(Q) Q=30
W(Q) X ignore & proceed	

Assume $T_1 \rightarrow W(Q) \rightarrow Q=20$

$T_2 \rightarrow W(Q) \rightarrow Q=30$

$TS(T_1)=1$ $TS(T_2)=2$ $TS(T_3)=3$

eg:



T_1	T_2	T_3
R(Q)		
W(Q)		
	W(Q)	
		W(Q)

IR ✓
~~FW~~ ✓
Inter R X
VS ✓

LOCK BASED PROTOCOL

In lock based protocol, a transaction applies lock on the data item before performing any operation on it & then unlocks it after completion of the operations.

Types of locks:

	shared	exclusive
shared	T	F
exclusive	F	F

1) Shared lock Lock-S(Q)
(Read-only lock)
- In this lock, data items can only be read.
- It can be shared among the transactions i.e. if a transaction applies shared lock on any data item, other transactions may also apply the shared lock on the same data item.

2) Exclusive lock Lock-X(Q)
- In this lock, data items can be read as well as written.
- This lock can't be applied by multiple transaction on same data item. Only one transaction can apply exclusive lock on any data item at a time

example

T_1	T_2
Lock-X(A)	
R(A)	
W(A)	
unlock(A)	
	Lock S(B)
	R(B)
	unlock(B)
Lock X(B)	
R(B)	
W(B)	
unlock(B)	
	Lock S(A)
	R(A)
	unlock(A)

- It doesn't ensure CS, VS, recoverability or cascadeless rollback.
- If data item is unlocked before committing, it may lead to inconsistency.
If data item is unlocked only after committing, concurrency is affected.
- May not be free from deadlock, starvation

infinite wait long wait

deadlock

T_1	T_2
X(A)	
	X(B)
X(B)	
	X(A)

T_1	T_2	T_3	T_4
X(A)	S(A)		
U(A)		S(A)	SA)
	U(A)		U(A)

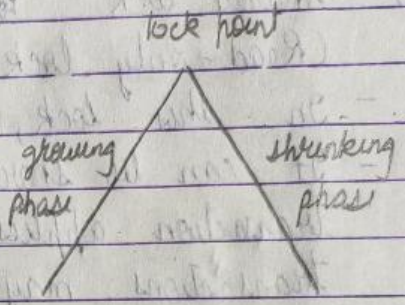
Due to these disadv., transaction must follow some set of rules for locking & unlocking data item eg 2PL, graph based.

2-PHASE LOCKING / Basic 2PL

In 2 phase locking protocol, each transaction in a schedule has 2 phases:

1) growing phase: In this phase transaction can only obtain locks but can't release any lock.

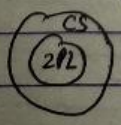
2) shrinking phase: In this phase, transaction can only release locks but can't obtain any lock.



- So transaction first enters into the growing phase, ~~then~~ obtains all the locks, then reaches the lock point & finally moves to shrinking phase & releases all the acquired locks
- It can perform read, write operation in both the phases.

Properties:

- Ensures CS & VS
- Order of serializability is the order in which transactions reaches lock point.
- May generate unrecoverable schedule or cascading rollbacks.
- Do not ensure freedom from deadlock/starvation
- Every 2-phase locking-protocol-schedule is CS but every CSS may not be generated from 2-Phase locking protol.



T1	T2
lock S(A)	
	Lock S(A)
lock X(B)	
unlock X(A)	
	Lock X(D)
unlock (B)	
	Unlock (A)
	unlock (D)

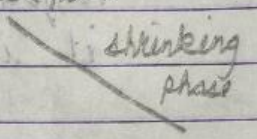
T_1	T_2
lock S(A)	
R(A)	
	lock X(B)
	R(B)
	W(B)
lock S(B) waiting	
	lock X(A) waiting

Deadlock

deadlock occurs when

Conservative / static 2PL — resolves deadlock problem

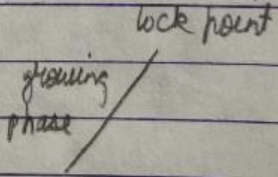
- In this protocol, there is no growing phase, transaction first acquires all the locks & then starts from the lock point and shrinking phase continues as usual.
- If all locks required by a transaction are not available, transaction must release all the locks aquired & wait.
- Prior knowledge of all the data items required is necessary



- It ensures CS, VS
- Independent from deadlock.
- Possibility of irrecoverable schedules & cascading rollbacks.

Rigorous 2PL — resolves irrecoverability & cascading rollbacks

In this protocol, there is no shrinking phase, transactions holds the lock until it is committed.



- ensures CS & VS
- ensures recoverability & cascadeless rollbacks.
- suffer from deadlock
- inefficient

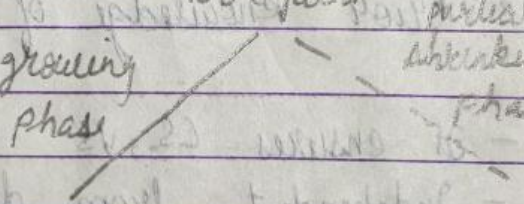
T1	T2
X(A)	
R(A)	
W(A)	
C	
U(A)	
	S(A)
	R(A)
	U(A)

Strict 2PL

improvement over rigorous 2PL

- In this protocol, partial shrinking phase is present.
- In shrinking phase unlocking of exclusive locks are not allowed but unlocking of shared locks can be done as they provide only read permission which do not updat the data item hence dirty read doesn't occur.

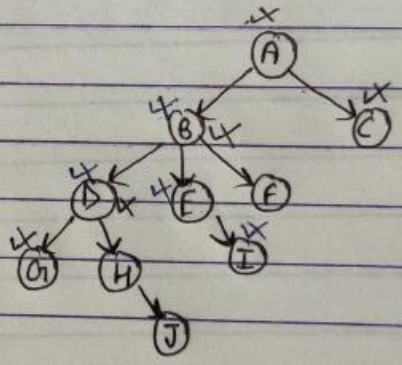
- All properties are same as rigorous 2PL except that it is efficient



GRAPH BASED PROTOCOL

- Graph based protocols require prior knowledge about the transactions.
- The order in which the database items are will be accessed must be known.
- Partial ordering is imposed on the data items ($\rightarrow$)
  $D \sim \{d_1, d_2, \ldots, d_n\}$
  if $d_i \rightarrow d_j$, then any transaction accessing both d_i & d_j must access d_i before d_j.
- Partial ordering may be applied through logical or physical organisation (eg linked list, ~~graph~~)
- After partial ordering, set of all data items D will be viewed as directed acyclic graph called database graph.
- Assumptions:
  graph must be rooted tree
  only exclusive lock is considered

Tree Protocol



Following rules must be applied
- First lock by T_i may be applied on any data item.
- Subsequently, a data item Q can only be locked by T_i if the parent of Q is currently locked by T_i.
- Data item may be unlocked at any time.
- Data item Q that has been locked & unlocked by T_i can't be relocked by T_i

T_1	T_2
lock X(B)	lock X(A)
lock X(D)	lock X(B)
lock X(E)	lock X(C)
unlock(B)	unlock(A)
lock X(I)	lock X(D)
unlock(D)	unlock(B)
unlock(E)	lock X(G)
unlock(I)	unlock(D)
	unlock(G)
	unlock(C)

- CS & VS ✓
- freedom from deadlock
- recoverability & cascadelessness not ensured.
- data item is unlocked as soon as it has been used → waiting time ↓ & concurrency ↑

- A transaction may have to lock data items that it doesn't access to access the desired data item
- leads to overhead waiting time & dec. in concurrency
- Transaction must know exactly what data item are to be accessed.

- Recoverability & cascadelessness can be attained by not unlocking before commit

Question :

Time Stamp Protocol

100 200 300

T_1 T_2 T_3

$R(A)^{100>0}$

$R(B)^{200>0}$

$W(C)^{100>0\ R}_{100>0\ W}$

$R(B)^{300>0}$

$R(C)^{100=100}$

$W(B)^{200<300}$ ← rollback

$W(A)^{300>100}_{300>0}$

	A	B	C
RTS	~~100~~	~~200~~ 300	~~100~~
WTS	~~300~~	0	~~100~~

↓

	A	B	C
RTS	100	300	100
WTS	300	0	100

T_1 & T_3 are executed successfully & T_2 will rollback.

Multi-Version Concurrency Control

- In this technique, multiple versions of an data item are maintained and values of data item are maintained when the item is updated.
- When transaction requires access to an item, an appropriate version is chosen to maintain the serializability of the schedule
- Due to this some read operations, that would be rejected in other techniques, can still be accepted by reading an older version of item to maintain serializability.
- Multi version technique requires more storage.
- But older versions may have to be maintained for recovery etc.

Multiversion Technique based on timestamp ordering:

In this method, several versions of each data item is maintained & for each version Q_i following data is recorded:

- Value of version Q_i
- Read TS (Q_i) - largest TS of trans. that read Q_i
- Writ TS (Q_i) - largest TS ——— writ Q_i

Rules:

If T_i issues read(Q_i) request

For read (Q), the recent version of Q is provided if ~~TS(Q_i)~~ TS(T_i) $\geq$ RTS(Q_i)

Database Recovery

Database systems are subject to failures
- Failures include
- system failure
- transaction errors,
- viruses
- catastrophic failure
- power failure etc.

A database system must be capable of recovering from these failures ensuring the transaction properties like atomicity, durability etc.

Recovery techniques are dependent on system log

System log: It is a special file which contains all the information about a transaction - start of transaction, end of transaction & all the operations performed by the transaction.
- This information is important for recovery.

System log contains following entries:

- < transaction T, start > : This entry records that transaction T has started the execution.

- < Transaction T, data item X, new_value > : This entry depicts that transaction T has overwritten the value of X to the new value.

- < Transaction T, data item X, old_value, new_value >: This entry depicts that transaction T has changed value of X from old_value to new_value.
  old_value - image of X
  new_value - afterimage of X

- < Transaction T, Commit > : This entry depicts that transaction T has completed all the operations & committed successfully.
- < Transaction T, abort > : abort
- Checkpoint : At this point all the previous logs are removed from the system & stored permanently on storage disk.

Undoing : If transactions are undone, log entry < Transaction T, data_item X, old_value, new_value > is used & value of data item X is set to old-value.

redo : If transactions are redone, log entry < Transaction T, data_item X, new_value > or < Transaction T, data_item X, old_value, new_value > is used & value of X is set to new_value.

Deferred Database Modification

- It is a log based recovery technique.
- In deferred update technique, database on disk is not updated until commit point.
- ∴ if a failure occurs before commit, changes are not made into the database ∴ UNDO is not required.
- But if failure occurs after commit, REDO is required.

Example :

T_1
R(A) 100
A = A + 100 200
W(A) 200
R(B) 200
B = B + 200 400
W(B) 400
Commit

A = 100
B = 200

initial values in database

A = 200
B = 400

after commit

log file
< T_1, start >
< T_1, A, 200 >
< T_1, B, 400 >
< T_1, commit >

Case : I Failure occurs before commit - no action

Case : II Failure occurs after commit - redo
A = 200 & B = 400 are stored

Immediate Database Modification

It is a log based recovery technique
In immediate update technique, changes are made permanent onto the disk as soon as operation is executed.
∴ If transaction fails before commit, undo is required.
If transaction fails after commit, redo is required.

Example :

T_1	
R(A) 100	A = 100, B = 200
A = A + 100 200	↓ W(A)
W(A) 200	A = 200, B = 200
R(B) 200	
B = B + 200 400	↓ W(B)
W(B) 400	A = 200, B = 400
Commit	

log file
< T_1, start >
< T_1, A, 100, 200 >
< T_1, B, 200, 400 >
< T_1, Commit >

Case : I Failure occurs before commit - undo
A = 100 B = 200 are stored.

Case : II Failure occurs after commit - redo
A = 200 B = 400 are stored.