31.10.2023

Book:

Backend - Database use ಮಾಡಿ

* Database: Huge amount of interrelated data collection which is ease to retrieve and having very good speed for access and search.

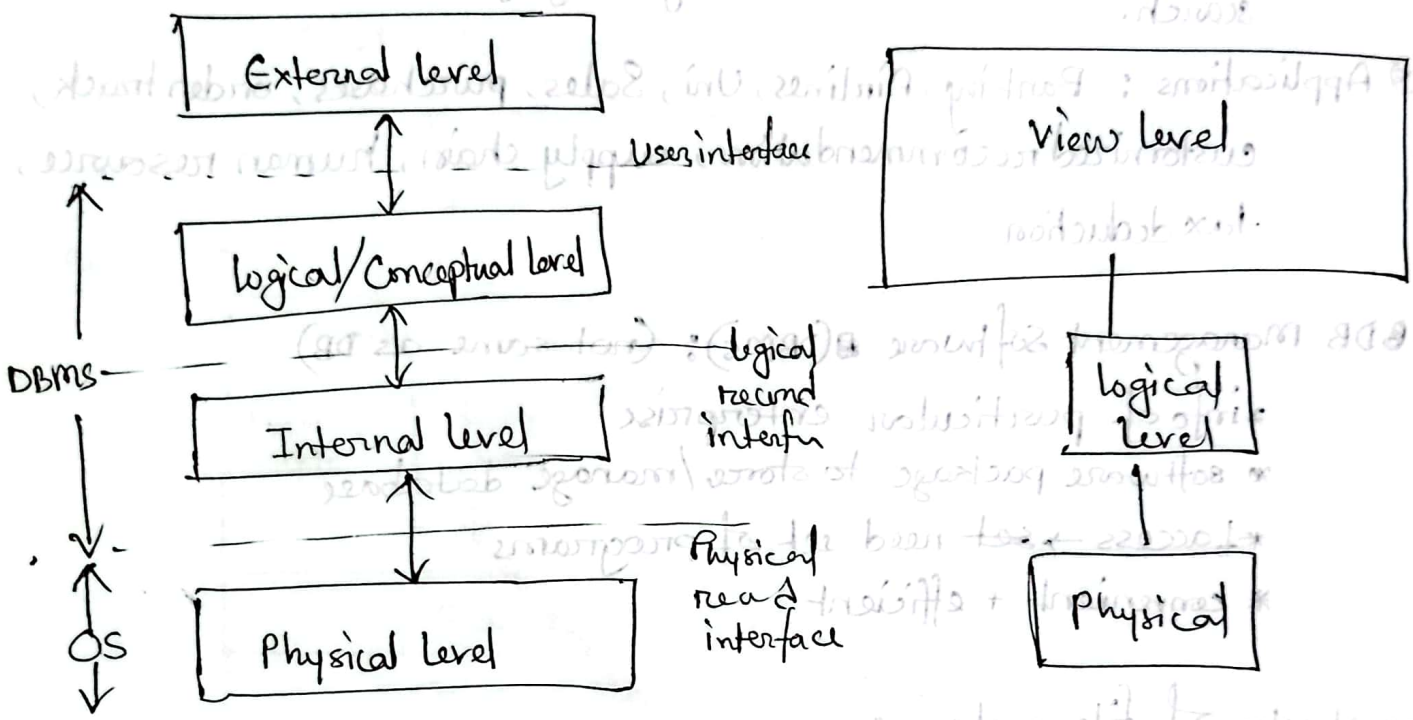
* Applications: Banking, Airlines, Uni, Sales, purchases, order track, customized recommendations, supply chain, human resource, tax deduction.

* DB Management Software (DBMS): (not same as DB)
  * info of particular enterprise
  * software package to store/manage database
  * to access → need set of programs
  * convenient + efficient

Drawbacks of file systems:
* data redundancy + inconsistency
* difficulty in accessing data
* data isolation
* integrity problems - (integrity constraints + hard to add constraints).
* atomicity of updates
* concurrent access by multiple users.
  → concurrent access needed for performance
  → uncontrolled concurrent accesses can lead - inconsistencies
* security problems → hard to provide user access to some but not all data

Abstraction levels

1. Physical level (record store description)
2. Logical level (data stored in DB + relationships among data)
3. View level (hides details of data types, views hide info for security)



Instances:

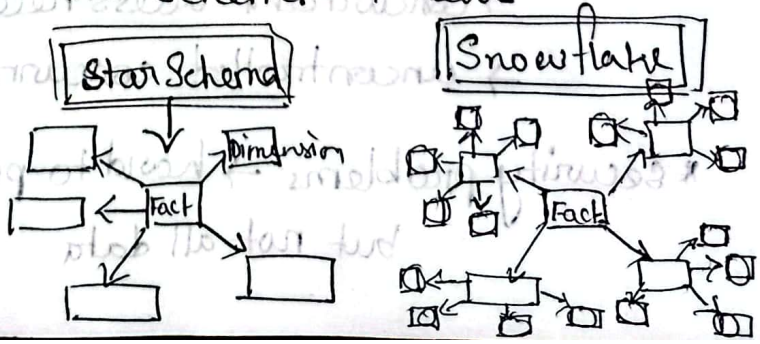
the actual DB applications
content of the database at a particular point in time

Schemas: overall design of DB

physical schema → overall physical structure of DB

logical schema → overall logical structure
[set of info + relationships]

sub schema → views

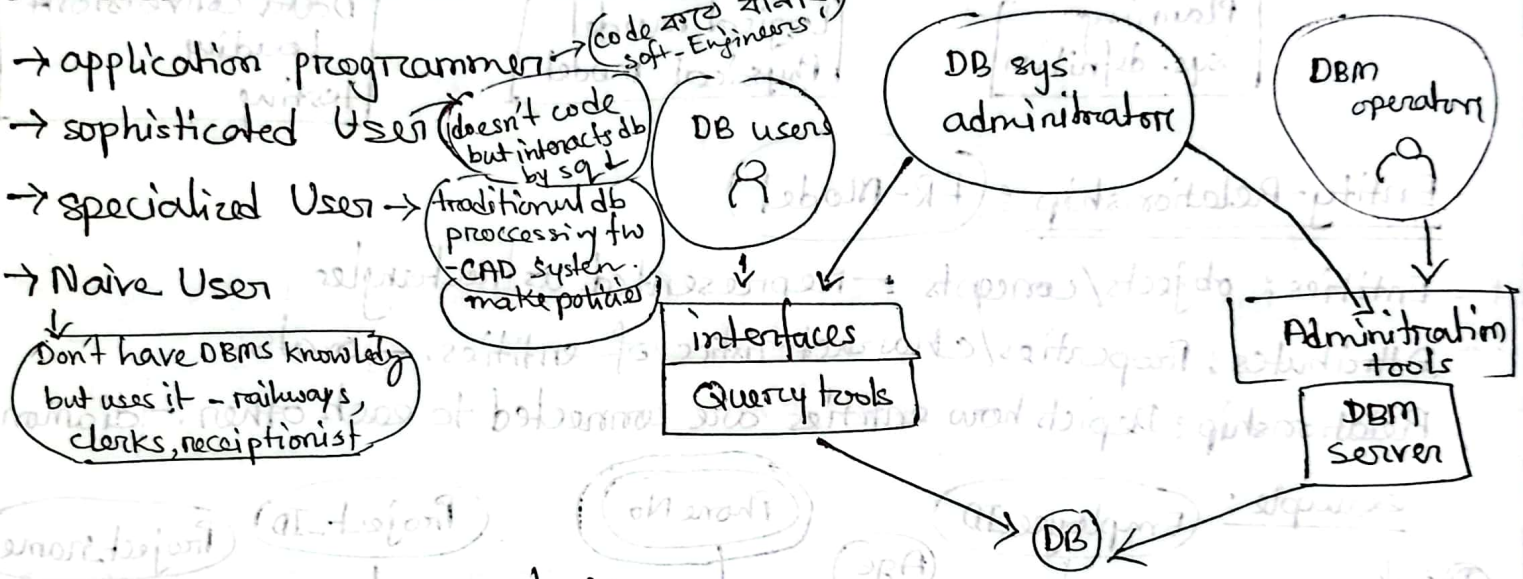


Data Independence (DI).

application that ensures the protection of data or structure is stored.

* **physical DI** : ability to modify physical schema without changing logical schema. [application depends on logical schema]
* **logical DI**: ability to modify logical schema wo changing view level.

Database Users & DBA Activities.



4 types SQL commands :

DDL (Data Definition Lang)
- CREATE (বানানোর সময়)
- ALTER (change করতে চাইলে)
- DROP (already আছে use করতে চাই না)
- RENAME (name change)
- TRUNCATE (table reset করবে)
- COMMENT

DML (Data Manipulation Lang)
- SELECT
- INSERT
- UPDATE
- DELETE
- MERGE
- CALL
- EXPLAIN PLAN
- Lock TABLE

DCL (Data Control Lang)
- GRANT (to give permission)
- REVOKE (to remove ,,)

TCL (Transaction Control Lang)
- COMMIT
- ROLLBACK
- SAVEPOINT
- SET TRANSACTION

2.11.2023

* Generate Project idea –
* Initial stakeholder meetings – ~~custo~~ clients, users etc
* Conceptualization – achievement + overall scope
* High level Description – vision statement that outlines purpose, goals, benefits.
  reference point for all parties to ensure alignment

Steps :

Requirements analysis → Planning sys. defination

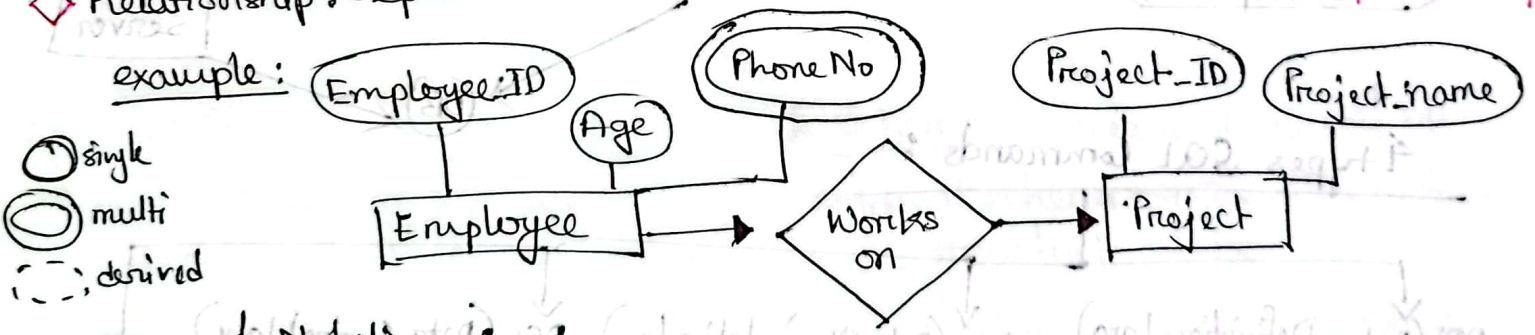
Database Designing → Logical model. Physical model

Implemention → Data conversion loading testing

Entity-Relationship : ER-Model

- Entities : objects/concepts – represented as rectangles.
- Attributes : Properties/characteristics of entities. – ovals.
- Relationship : Depicts how entities are connected to each other. – diamonds

example :



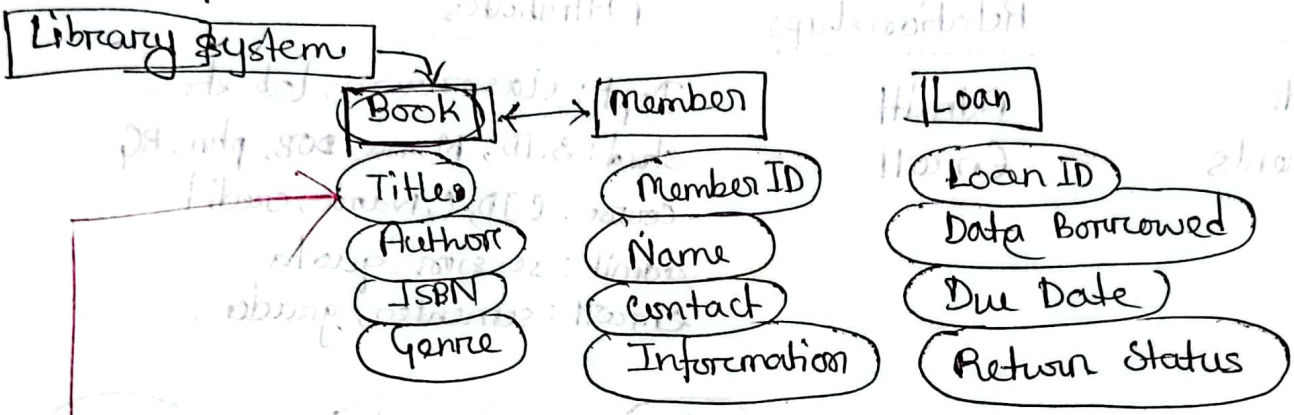
Types of Notations :

1. Crow's Foot Notation
2. Chen Notation
3. Barker's Notation
4. UML Class Diagrams

* entity connect হয়ে চাইলে relationship নাও
* relationship basically a table ~~that~~ takes attributes from the associate tables and may add more attributes if needed.

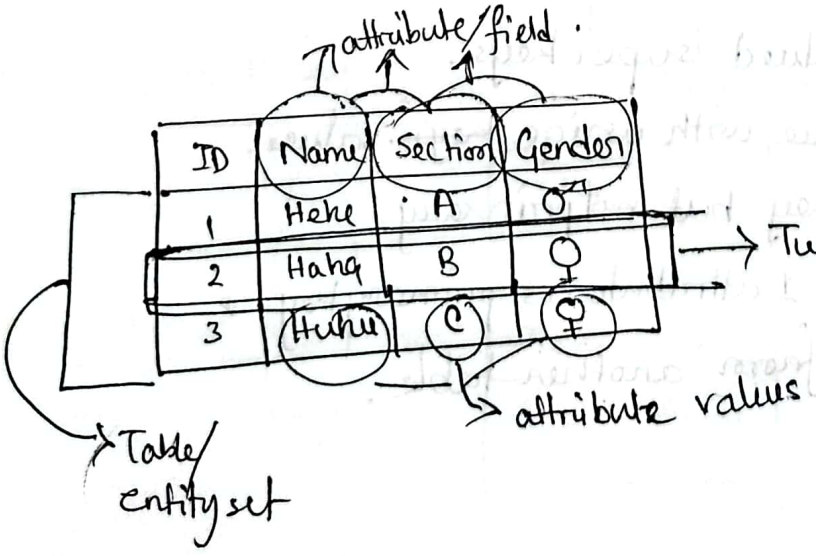
Entity: all time singular noun concise.
requirement analysis - এ যত্তো entity - relationship খুজি
→ real obj from prob Domain. + distinguishable

people - employee, students, patients.
place - store, warehouse.
obj - machine, products, vehicle
events - lectures, sales, registration.
concept - account, course



Relationship: * Connecting entities
* Defining Business Rules
* Cardinality (connection of instances of entities).
* Attributes
→ Borrowing book expl.

Attributes: * properties
* provide details + help define/differentiate
* relevant and necessary ones should be selected.

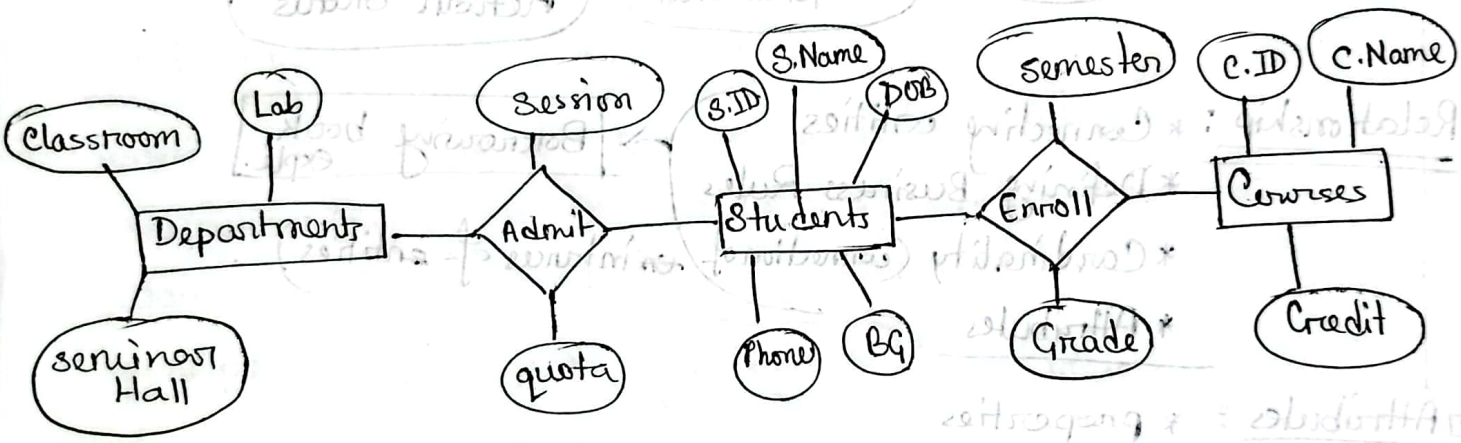


7.11.2023

ERD example:

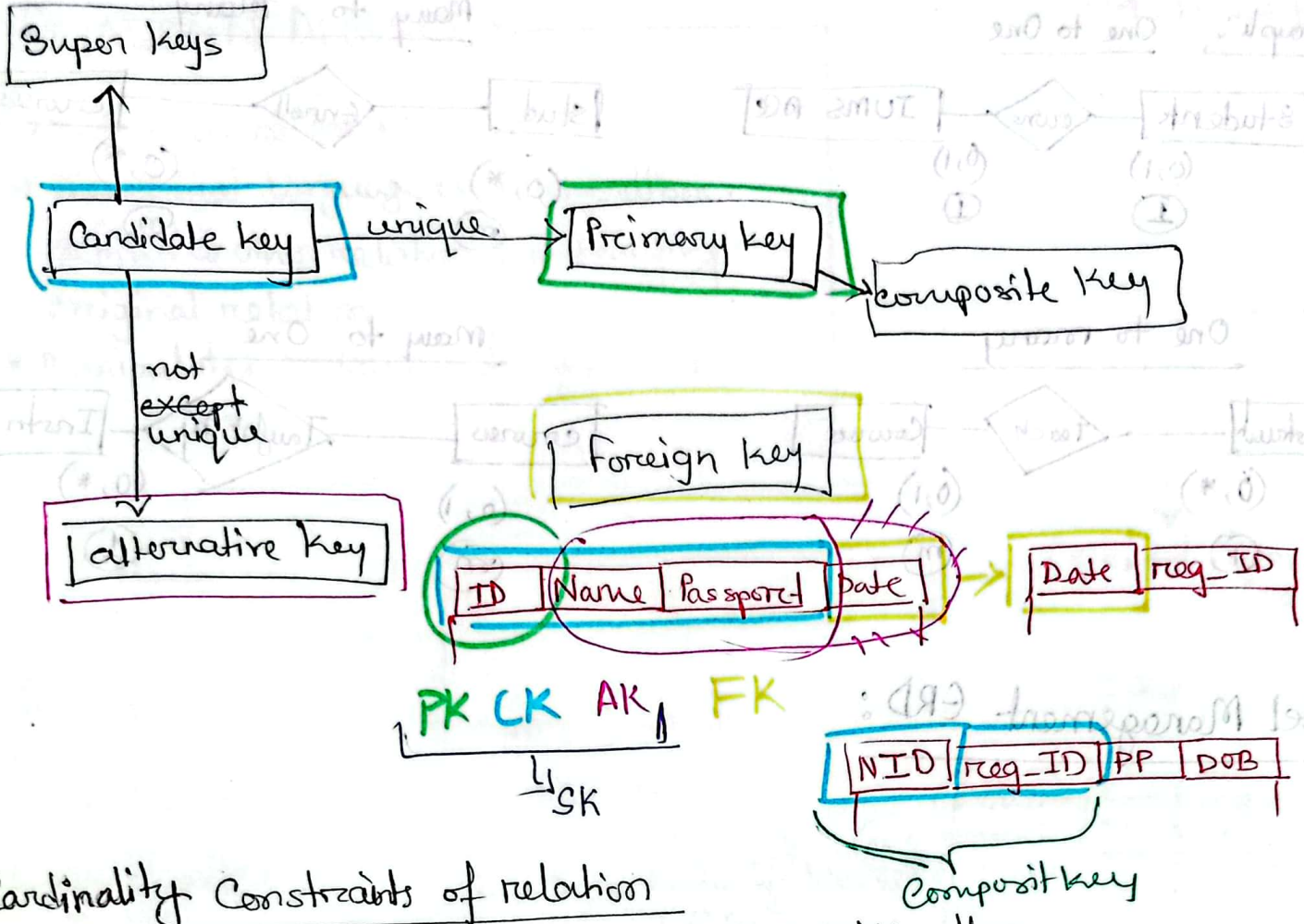
There are 8 departments in aust. Students are admitted in different departments. Each of the department has classrooms, labs, etc. Students are enrolled in different courses in a specific semester and receive grades.

Entity	Relationships	Attributes
Students	Admitt	Dept: classroom, lab etc.
Departments	Enroll	stud: S.ID, Name, DOB, phn, BG
Courses		course: C.ID, C.Name, credit
		admit: session, quota
		enroll: semester, grades.



Keys:

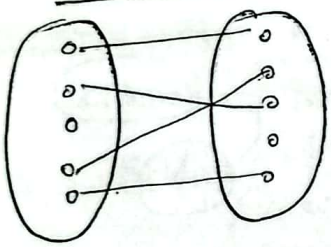
1. Super keys: all attributes
2. Candidate keys: minimal valued super keys.
3. Primary keys: candidate value with unique ~~keys~~ values.
4. Alternative key: candidate key but not primary.
5. Composite key: more than 1 attribute as primary key.
6. Foreign key: primary key from another table.



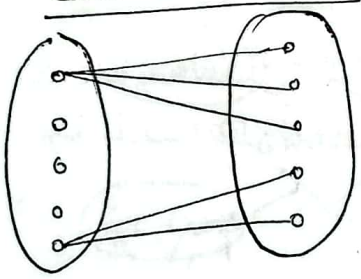
Cardinality Constraints of relation

express max. no. of entity that associates with other.

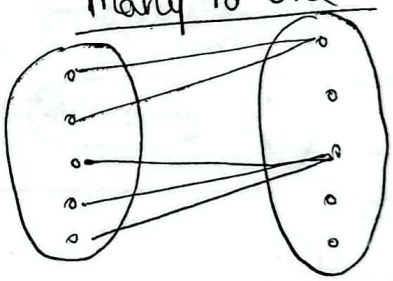
One to one



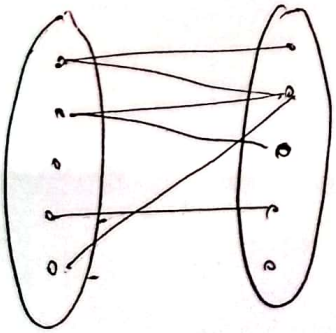
One to Many

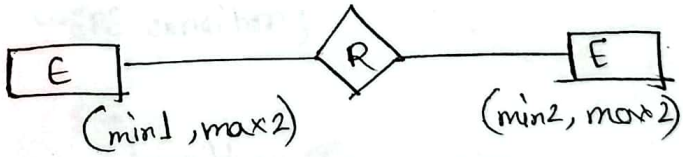


Many to One



Many to Many

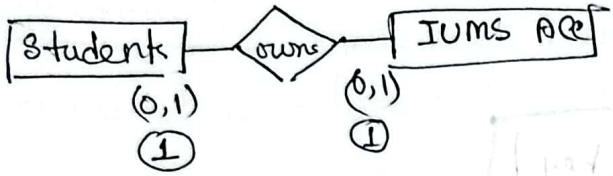




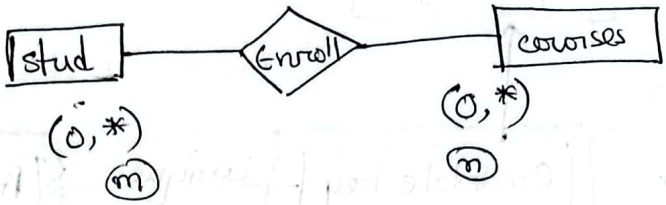
R	(1)	(2)
Many - Many	(0, *)	(0, *)
Many - One	(0, 1)	(0, *)
One - Many	(0, *)	(0, 1)
One to One	(0, 1)	(0, 1)

shortcut
mm (0,*)
11 (0,1)
m1 (0,1) (0,*)
1m (0,* (0,1

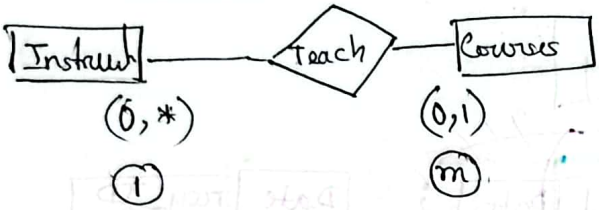
Example: One to One



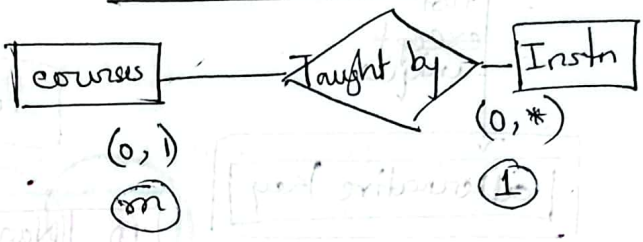
Many to Many



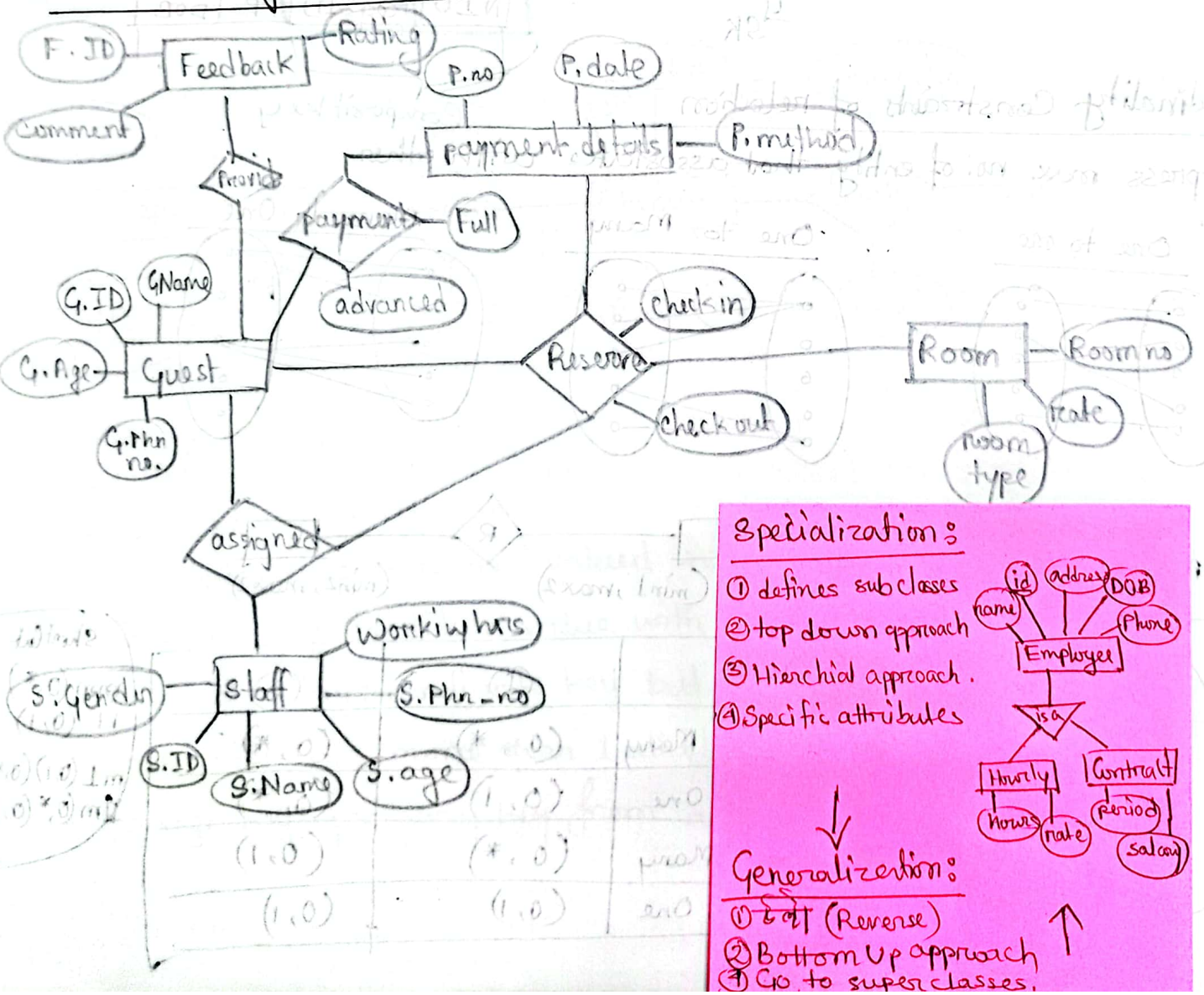
One to many



Many to One



✱Hotel Management ERD:



Specialization:

① defines sub classes
② top down approach
③ Hierchial approach.
④ Specific attributes

Generalization:

① Bottom Up (Reverse)
② Bottom Up approach
③ Go to superclasses.

14.11.23

RELATIONAL ALGEBRA :

* Mathematical forem
* Theoretical Language with operations. defines another relation wo changing original relation.
* Manipulates relations to get result

- Basic
  - UNARY [π σ ρ]
    - Projection (π)
    - Selection (σ)
    - Rename (ρ)
  - BINARY [$\cup$ $\times$ $-$]
    - Union ($\cup$)
    - Cross Product/Cateisian ($\times$)
    - Minus/Difference ($-$)
- Extended /Derived [$\bowtie$ / $\cap$]
  - Join ($\bowtie$)
  - Division ($/$ or $\div$)
  - Intersection ($\cap$)

SELECTION : * conditional statement -এ কাজ করে
* selecting specific row.

syntax → $\sigma_{condition}(R)$
symbols → $<>, \neq, \leq, \geq, \wedge, \vee, \neg$
SQL → SELET * FROM table WHERE condition;

Example: Table name → Employee.

* show employee whose age less than 30 : $\sigma_{age<30}(Employee)$.
* " " age more than ~~50~~ 25 and salary is more than 5000/- : $\sigma_{age<25 \wedge salary>5000}(Employee)$
* " " age below 25 or higher 40 : $\sigma_{age<25 \vee age>40}(Employee)$
* " " name starts with "mar" : $\sigma_{name = "mar\%"}(Employee)$.
  - SQL : SELECT * FROM Employee WHERE Name like "mar%";

PROJECTION : * shows specific result.
* works on 11 columns.

syntax → $\Pi_{attributes}(R)$
SQL → SELECT col1, col2 FROM table;

Examples: Table name = Staff
1. Name of staffs : $\Pi_{name}(Staff)$
2. show names and salary : $\Pi_{name, salary}(Staff)$
3. 11 details of staffs whose salary is more than 5000/- : $\Pi_{name, salary, age}(\sigma_{salary>5000}(Staff))$.

→ SQL : SELECT * FROM Staff where Salary > 5000;

~~RENAME~~

RENAME :
* tables are not compatible.
* When need to compare same relation.

syntax → $\rho_{old\ name \to new\ name}(R)$

SQL → SELECT col1 AS c1, col2 AS c2 FROM table;

Examples :

Employee		
Name	Branch	Salary

change branch to location, salary to pay : $\rho_{branch, salary \to location, pay}(Employee)$

SQL:
SELECT Branch AS Location, Salary AS Pay FROM Employee

15.11.23

BINARY Operation COMPATIBILITY.

*must be union compatible → no. of attributes same.
no. of colorumn same
Domain type same.

Example:

X	
A	B

Y	
C	D

not compatible

X	
A	B

Y	
A	B

compatible.

UNION: সব নিব

R	
A	B
α	1
α	2
β	1

S	
A	B
α	2
β	3

→ RUS =

RUS	
A	B
α	1
α	2
β	1
β	3

INTERSECTION: Common data নিব

- ~~RU~~ R∩S =

R∩S	
A	B
α	2

DIFFERENCE: সামনেরটা থেকে পরের data বাদ দিব

R-S =

R-S	
A	B
α	1
β	1

S-R =

S-R	
A	B
β	3

DIVISION: uncommon columns বাদ দিব → ABC/C = AB.
remainder বাদ
Common elements with dividend = quotient,
" " না. হলে = 0 "

R	
A	B
α	1
α	2
β	1
β	2

S
B
1
2

R/S =

R/S
A
α

R	
A	B
S1	P1
S1	P2
S1	P3
S1	P4
S2	P1
S2	P2
S3	P2
S4	P2
S4	P4

/

S
B
P2

=

R/S
A
S1
S2
S3
S4

R/

T
B
P4

=

R/S
A
S1
S4

ASSIGNMEN OPERATION:

Temp1 ← Borrower X Loan

Temp2 ← Borrow. Loan_no = Loan. Loan-no.

Temp3 ← Branch = "Gulshan"

$\Pi_{name}(\sigma_{Temp2 \wedge Temp3}(Temp1))$

CARTESIAN PRODUCT :

R
A
α
β

S
B
1
2
3

R x S =

R x S	
A	B
α	1
α	2
α	3
β	1
β	2
β	3

R	
A	B
α	1
α	2

S	
B	C
1	X
2	Y

R x S =

R x S			
A	R.B	S.B	C
α	1	1	X
α	1	2	Y
α	2	1	X
α	2	2	Y

$\beta_{B \to R.B}(R)$

$\beta_{B \to S.B}(S)$

CARTESIAN PRODUCT

1. Small Data set
2. less columns
3. No common / one maybe common columns.
4. Row returns unused.
5. Same named columns needs to get renamed.

JOIN OPERATION

1. Large data set
2. Many columns.
3. At least one common column.
4. Useful row returns.
5. No need to rename.

CARTESIAN EXAMPLE:

Borrower	
name	loan no.
A	L-17
B	L-23

Loan		
Loan-no	Branch	Amount
L-17	Banani	75,000
L-23	Gulshan	50,000

B × L				
name	loan no	Loan no	branch	Amount
A	L-17	L-17	Ban—	75K
A	L-17	L-23	Gul—	50K
B	L-23	L-17	B—	75K
B	L-23	L-23	G—	50K

* Find customers of bank who have loan at GULSHAN.

- Rename

$\rho_{Loan\_no \rightarrow B.Loan\_no}(Borrower)$

$\rho_{Loan\_no \rightarrow Loan.Loan\_no}(Loan)$.

- Product

Borrower × Loan = B × L

Rel. Algebra: $\Pi_{name}(\sigma_{B.Loan\_no = Loan.Loan\_no \wedge branch = "Gulshan"}(B \times L))$

JOIN OPERATION

- NATURAL
- LEFT
- RIGHT
- FULL

— OUTER JOIN

NATURAL:

A	
name	dept
A	CSE
B	EEE
C	EEE

B	
dept	name
EEE	D
CSE	E
ME	F

A ⋈ B =

A ⋈ B		
name	dept	name
A	CSE	E
B	EEE	D
C	EEE	D

* Natural Join drawbacks : row is lost (dangling tuples).

* Borrower loan – এর example – use করব

$\Pi_{name}(\sigma_{branch = "Gulshan"}(Borrower \bowtie Loan))$.

THETA JOIN : Joins with comparison Operator.

$(R \bowtie S)_\theta$

or, $\sigma_{condition}(R \bowtie S)$

$\Pi_{name}(Depositor \bowtie Account)_{Balance > 5000}$.

LEFT JOIN : * priority → left table.
*value not matched – right table – এ NULL বসাব।

Faculty	
name	Dept
Joti	CSE
Oyishee	EEE
Neha	BBA

Head	
dept	name
ME	Rashed
CSE	Alvi
EEE	Ziyad

F ⋈ H

F ⋈ H		
name	dept	name
Joti	CSE	Alvi
Oyishee	EEE	Ziyad
Neha	BBA	NULL

RIGHT JOIN: * prior → right table.

F ⋈ H

F ⋈ H		
name	dept	name
NULL	ME	Rashed
Joti	CSE	Alvi
Oyishee	EEE	Ziyad

FULL OUTER : * combine left + right

F ⟗ H

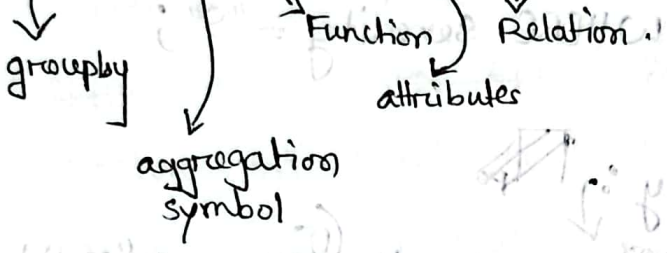
name	dept	name
Joti	CSE	Alvi
Oyishee	EEE	Ziyad
Neha	BBA	NULL
NULL	ME	Rashed

AGGREGATE FUNCTION:

* take a collection of values
* return a single value.
* attribute same (of value).

Functions:
- Max
- Min
- Sum
- Count
- Count - distinct
- Average
- Etc.

Syntax: $\mathcal{G}_{groupby}\ G_{F(A)}\ (R)$



Avg-median

n = 1 2 3 4 (5) 6 7 9 11

↓

Median [non changable]

avg = $\frac{\Sigma n}{9}$ [changable]

Example:

* Find total salary of instructors : $G_{sum(salary)}(Instructor)$
* total number of instructors : $G_{count(ID)}(Instructor)$.
* " " " dept of " : $G_{count\text{-}distinct(dept)}(Instructor)$.
* avg salary of instructors at dept : Department $G_{avg(salary)}(Instructors)$

sql: SELECT dept., AVG(Salary) AS avg-salary
FROM Instructor GROUP BY dept

Practice (from lec-)

① Docs don't practice surgery : $\Pi_{DoctorID}(\sigma_{speciality \neq "Surgery"}(Doctors))$

SQL → SELECT ~~* FROM~~ DoctorID FROM Doctors
WHERE speciality <> "Surgery";

② Severity 'S' patients with their id and ward code : $\Pi_{PatientId, WardCode}(\sigma_{Severity = "S"}(Patient))$

SQL → SELECT PatientId, WardCode FROM Patient
WHERE Severity = "S";

③ Info of all patient who got admitted in "Medicine" category :

$\Pi_{PatientId, PatientName, WardCode, Severity}(\sigma_{category = "Medicine"}(Patient \bowtie Admission))$

SQL → SELECT * FROM Patient LEFT JOIN Admission ON
Patient.PatientId = Admission.PatientId AND
Patient.Category = "Medicine";

④ Docs whose salary more than 45000 :

$\Pi_{DoctorId, FirstName, LastName, Speciality, ServiceYear, Salary}(\sigma_{Salary > 45000}(Doctors))$

SQL → SELECT * FROM Doctors WHERE Salary > 45000;

⑤ Avg bill for patient : $\mathcal{G}_{AVG(BILL)}(Admission)$

SQL → SELECT * FROM Admission HAVING AVG(BILL)

⑥ show doc of patient Id = 6002 :

$\Pi_{FirstName, LastName}(\sigma_{PatientId = 6002}(Doctors \bowtie Admission))$

SQL → SELECT * FirstName, LastName FROM Doctors ~~OUTE~~ FULL
OUTER JOIN Admission ON Doctors.DoctorID = Admission.DoctorID

WHERE PatientID = 6002 ;

⑦ patient + doc name where bill is not morce than 10 000:

$\Pi_{FirstName, LastName, PatientName}(\sigma_{Bill \leq 10\,000}(Patient \bowtie Admission \bowtie Doctor))$.

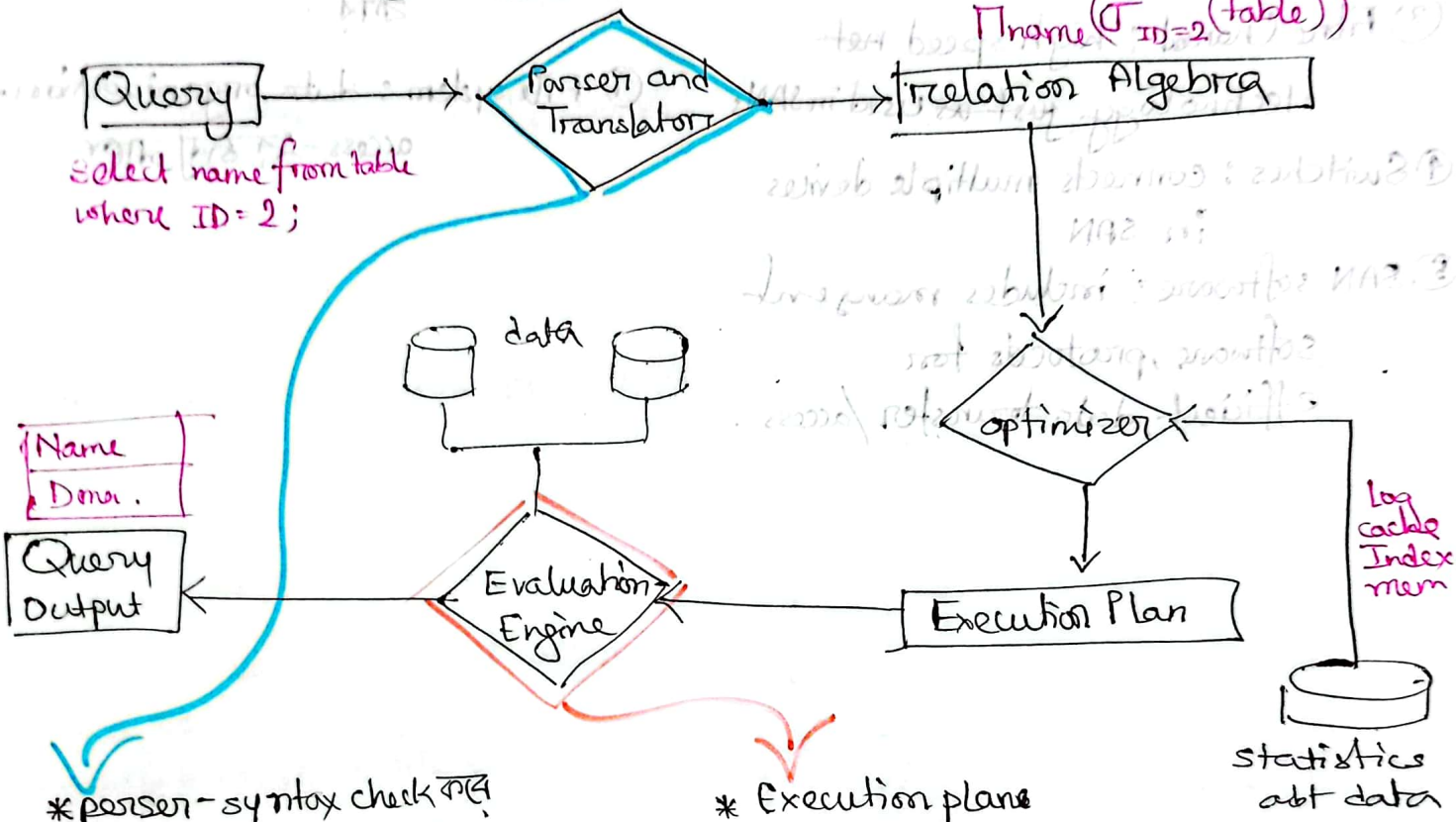
SQL :→ SELECT FirstName, LastName, PatientName FROM Patient
FULL OUTER JOIN Admission ON Patient.PatientID =
Admission.PatientID ~~UNION~~
~~SELECT FN, LN, PN FROM Admission OL~~ FULL OUTER JOIN
Doctore . ON Admission.DoctorId = Doctore.DoctorId
WHERE . Bill ≤ 10 000;

Query Processing

→ range of activities that extracts data from database.

Basic Steps:

① Parsing and Translation
② Optimization
③ Evaluation



* parser - syntax check করে relation verify করে
* translator - internal form-এ নেয় অর্থাৎ RA তে নেয়

* Execution plans consider করে
* output ready করে দেয় plan wise

⊞ Execution/Evaluation Plan : specifying detailed evaluation strategy ~~is called~~ .

Query Cost factors : disk access, CPU, even network communication

Disk access : ~~द्वारा~~ accountable : ① no. of seeks ~~द्वारा~~ → ④ avg no. of seeks cost
② no. of blocks read → ⑤ avg. no. of blks read cost
③ no. of blocks written ⑥ avg no. of blk written cost

Simply we use no. of block transfer and no. of seeks.

t_T = time to transfer 1block
t_S = time for one seek

Cost for b block transfers S seeks

$$(b \times t_T) + (S \times t_S)$$

Exercise: 125 blocks of code, requires 21 seeks to build.

1 seek time = ~~7ms~~ 11ms
1 block transfer time = 7ms

Cost = ?

Ans:

$t_s = 11ms$

$t_T = 7ms$

$b = 125$

$S = 21$

$$Cost = (b \times t_T) + (S \times t_s)$$

$$= (125 \times 7) + (21 \times 11) = 202125 \text{ ms}$$

$$= 202.125 \text{ s}$$

30. 11. 2023

Transaction Concept

↳ unit of program execution —— accesses + updates data [not monetary]

(*) deals with: ① Failure of various — hardware / system crashes.
② Concurrent execution of multiple transactions.

ACID properties:

→ Durability: After successful transaction — changes it has made persists even if system failure.

→ Isolation: Each transactions stays unaware of other concurrent executions (transactions).
Intermediate transaction results are hidden too.

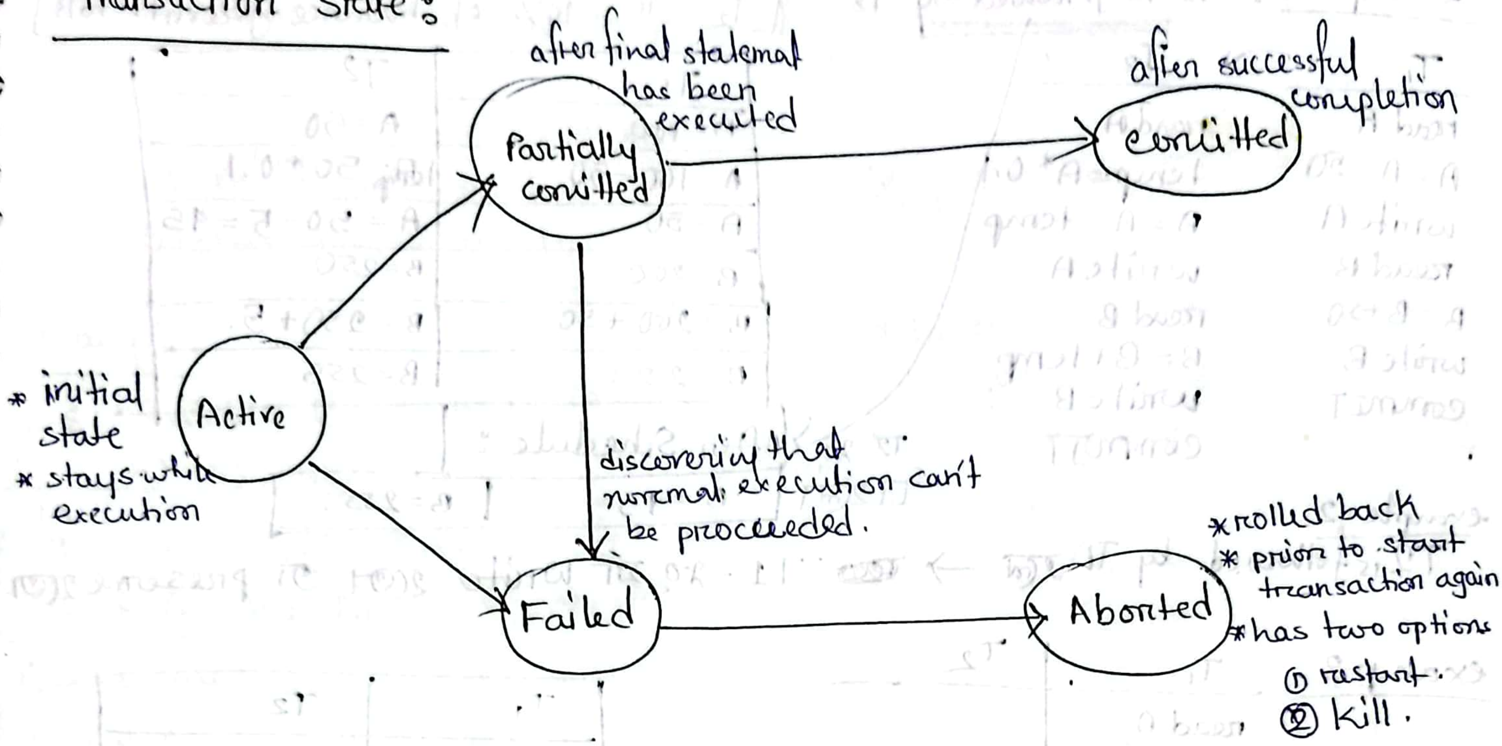
For every pair transactions T_i, T_j ; it appears to T_i that either T_j finished before T_i or T_j started after T_i

→ Consistency: Execution in isolation preserves consistency

requirements: ① sum of A, B is unchanged by transaction
② when started to execute database, transaction must see consistent DB.
③ during execution db must be TEMPORARILY INCONSISTENT.
④ After execution done db must be CONSISTENT
⑤ Erroneous transaction logic leads to inconsistency.

→ Atomicity: either all of the operations are reflected or none are.

Transaction state:



◻ **Concurrent Executions:** multiple transaction are allowed concurrently.

advantages → * increased processor and disk utilization.
better throughput

* reduced average response time
short ones don't need to wait behind long ones.

◻ **Concurrency control schemes:** mechanisms to achieve isolation.
* prevents interaction among CONCURRENT transactions so that the consistency persists of the database.

Schedule: sequences of inst. that specifies - chronological order in which instructions of concurrent transactions are executed.

must consist: ① all instruction of those transactions
② preserve the order.

* successful transaction-কে বলে 'commit' হলে [by default থাকে last statement হিসেবে থাকে]

* transaction failed = abort (rollback)

example 1 : T1 is followed by T2

[T_1 · transfer \$50 from A to B.
T_2 " 10% of balance from A to B]

T_1	T_2
read A	
A = A − 50	
write A	
read B	
B = B + 50	
write B	
COMMIT	
	read A
	temp = A * 0.1
	A = A − temp
	write A
	read B
	B = B + temp
	write B
	COMMIT

T_1	T2
A = 100	A = 50.
A = 100 − 50	temp = 50 * 0.1
A = 50	A = 50 − 5 = 45
B = 200	B = 250
B = 200 + 50	B = 250 + 5.
B = 250	B = 255.

T2 - এ শেষ হলো → After Schedule :

A = 45	B = 255.

example-2
T2 is followed by T1 হলে → T1 - এর মান write হলো তা preserve হলো

examp-3

T_1	T_2
read A	
A = A − 50	
write A	
	read A
	temp = A * 0.1
	A = A − temp
	write A
read B	
B = B + 50	
write B	
commit	
	read B
	B = B + temp
	write B
	commit

T_1	T2
A = 100	A = 50
A = 100 − 50	temp = 50 * 0.1
A = 50	A = 50 − 5 = 45
B = 200	B = 250
B = 200 + 50	B = 250 + 5
B = 250	B = 255
A = 45	B = 255

example-4 → not preserved.

T_1	T_2
read A	
A = A − 50	
	read A
	temp = A * 0.1
	A = A − temp
	write A
write - A	
read B	
B = B + 50	
write B	
Commit	
	read B
	B = B + temp
	write B
	commit

Serializability: transaction preserves db consistency (assumed)

↓

via serial execution.

① conflict serializability → (swap করে conflict serialized করা যায়)

② view serializability.

* belong to diff transactions
* operate on same data item.
* one of them should be write (at least)

example:

S1 : $R_1(A), W_1(A), R_2(A), W_2(A), R_1(B), W_1(B), R_2(B), W_2(B)$

Here,
T1 : $R_1(A), W_1(A), R_1(B), W_1(B)$
T2 : $R_2(A), W_2(A), R_2(B), W_2(B)$

possible schedules: T1 → T2 or T2 → T1

S11 : $R_1(A), W_1(A), R_1(B), W_2(B), R_2(A), W_1(B), R_2(B), W_2(B)$

S12 : $R_1(A), W_1(A), R_1(B), W_1(B), R_2(A), W_2(A), R_2(B), W_2(B)$

Exple: conflict : $(R_1(A), W_2(A))$
conflict pair : $(W_1(A), W_2(A))$
$(R_1(A), R_2(A))$.
non-conflict : $(R_1(A), W_2(B))$
" " : $(W_1(A), W_2(B))$

Conflicting Instructions:

✗ ① I_i = read (Q), I_j = read (Q) [don't conflict).
✓ ② $I_i = W(Q)$, $I_j = R(Q)$
✓ ③ $I_i = R(Q)$, $I_j = W(Q)$
✓ ④ $I_i = W(Q)$, $I_j = W(Q)$

T_3	T_4
read (Q)	
	write (Q)
~~read~~ write (Q)	

not conflict.

Precedence Graph/Serialization Graph:

used to test conflict serializability.

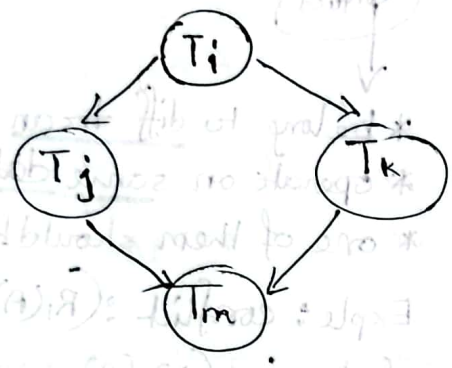
Algo: 1. create node T.
2. For, R(X) & W(X) → T_i এর W(X) - এর পর T_j R(X) execute করলে
→ then $T_i \rightarrow T_j$ edge আঁকব।
3. " W(X), R(X) → T_i এর R(X) - এর পর T_j W(X) execute করে
→ $T_i \rightarrow T_j$.
4. " W(X), W(X). if T_i - এর W(X) - এর পর T_j W(X) execute করে
→ $T_i \rightarrow T_j$

Acyclic graph ⟶ Conflict serializable

∴ Detect cycle → if exists ⟶ non serializable.

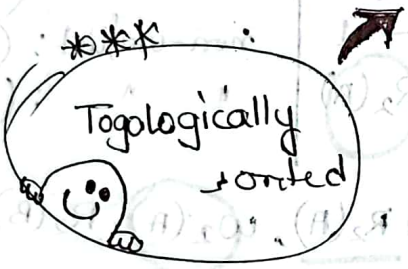
if acyclic ⟶ topologically sort करके serializability order मिलता है।

Expln :



can be :

or can be :



7.12.2023

NORMALIZATION :

* technique to organize the data in the dB.
* eliminate data redundancy (repetition).
  - insertion
  - updates
  - delete anomalies.
* ensures data dependencies

→ usage

Functional Dependency :

* set of constraints btwn 2 Attributes.
* if two tuples have same value for tuples – A1, A2 ... An then those two for B1, B2 .... Bn must be same too.
* X → Y (→) sign use करके :
  X functionally determines Y.
* Left attributes determine right ones.

Example :

Student ID → (First_Name, Last_name)

Course_no, course_name → Faculty_name

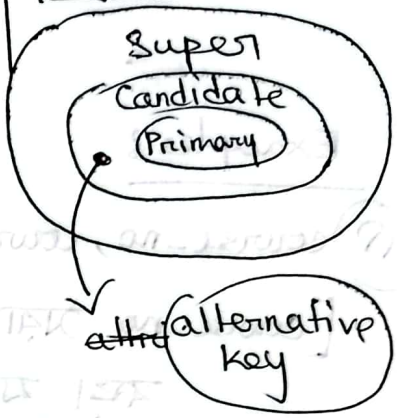
Course_no. → course_name, faculty_name.

marks → grade.

CANDIDATE KEY :

* identifies tuples
* might be more than 1
*

Keys :



attribute → alternative key

* Composite key – 2 या primary key – 2 या 3 joint attributes.
* Foreign key – दूसरी table – 2 primary key (to link tables)

***

Left – 2 या 3 और right – 2 या 3 determine कर रहे

⊞ Partial Functional Dependency:

* composite determinant থেকে attribute সরায় দিচ্ছে
* but dependency change হয়নি।

⊞ Full functional Dependency:

composite determinant থেকে attribute সরায় নিলে dependency change হয়ে যায়

Examples:

(PFD) course_no, course_name → Faculty_name

[course_no সরায় নিলেও course_name দিয়ে faculty detect করা যাবে]

(FFD) Student_ID, course_no → marks

[Student_ID সরাইলে → দুই course-এ সবাই same mark হয়ে যাবে]

⊞ Transitive Functional Dependency: chain create হয়

eg:
A → B
B → C
C → A

*Why Normalize?

- ⇒ extra memory exist
- ⇒ difficult to handle, update.
- ⇒ stops data inconsistency.
  - insert anomaly
  - delete "
  - update "

Forms:
1. First Normal Form (1NF)
2. Second " " (2NF)
3. Third " " (3NF).
4. Boyce & Codd Normal Form (BCNF)
5. Forth " " (4NF)
6. Fifth " " (5NF)
7. Sixth " " (6NF)

Steps to do:
1. specify Primary key
2. " FD. [determinist + obj of determ.]
3. appy rules of forms.
4. Keep chaging. → retesting + modifying to meets the conditions of each form.

12.12.2023

1NF : conditions :

* atomic attributes/combos (values - single)
* same domain in a column
* unique column names.
* order does not matter.

2NF : * should be in 1NF.

* no partial dependency. (determinant change করলে হবে না!)
* all need Fully fn dependent

3NF : * should be in ~~3NF~~ 2NF

* no transitive dependency.

BCNF : * should be in 3NF.

* A → B
  - must be a candidate key (superkey ও)
    - might be composite/overlapping
      - primary key is generated.

Indexing : speeds up access to desired data

* SEARCH KEY :- attributes to set of attributes used to look up records.
* INDEX FILE :- INDEX ENTRIES. (smaller than original file)

basic kinds
① → Ordered indices (stored in sorted order – search keys)
② → Hash indices (distributed uniformly across BUCKET using hash fn)

Evaluation Metrics:

* access type supported efficiently.
  - records specified value.
  - " " range of values.
* access TIME (lower = better ; retrival time).
* INSERTION Time (crucial).
* DELETION Time (faster = better).
* SPACE overhead (lower better ; impacts db size, additional mem ही)

ORDERED INDICES :

Primary / clustering index.
* search key usually but not necessarily primary key होती
* ही search key specifies order.

Secondary / non-clustering.
* search key specifies different from subsequential keys.

Dense Index (Files) : records appear for every search key

search keys
1014
1025
1037
4566
2988

1014	A	10
1025	B	11
1037	C	12
4566	D	45
2988	E	9

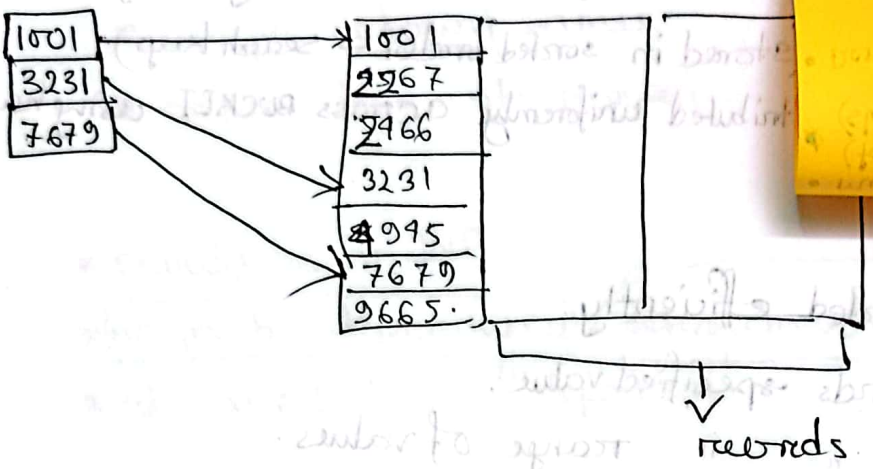
records.

Files
- Dense — fast
- Sparse — slow, less space, less maintanm.

Sparse Index (File)

has index records for only some search keys. [primary ordered file এর জন্য]

Locating:
* find index record — largest search key value.
* search file sequentially starting at record to which the index record points

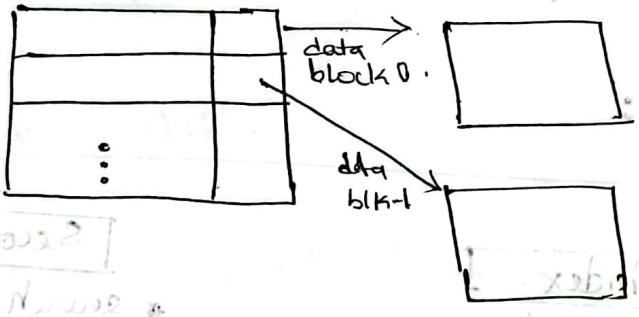


~~Dense~~ / ~~Vs~~ / ~~Sparse~~

~~* fast~~ * slow

Good tradeoff:

* index entry for every block in file.

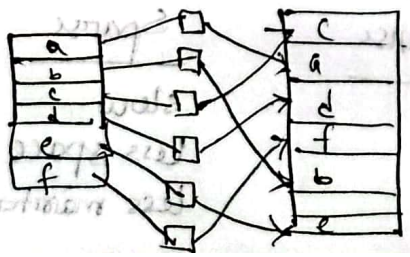


Secondary Indices:

(dense files use করে)

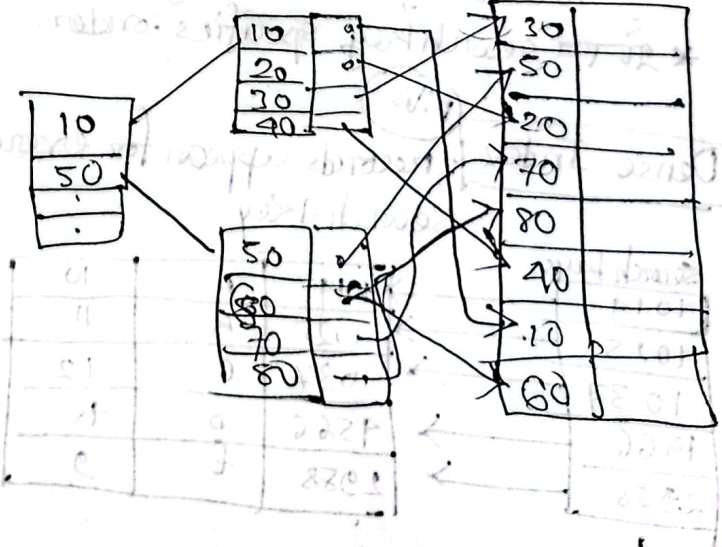
data can be found but not sorted.

also known as non-clustered



প্রত্যেক record এর জন্য

~~Primary Vs Secondary~~



Non cluster (2ndary)

- Search K → Candidate
- May or may not be sorted
- SK → not null
- More time (faster than clus) (slower " pri)
- Dense

Cluster

- Search K → may / not unique (unique হলে Pri)
- Sorted.
- SK → not null also
- faster (pri) slower (cluster) as pri → SK unique.
- Requires extra work for indexing.
- SK → non key values
- duplicate থাকে (না থাকলে pri)
- sparse

Binary vs Higher order Tree

Binary
* in memory search
* minimizes no. of mem-access

Higher order Tree
* searches data on block devices
* minimize no. of device access.

M-ary
* branching factor max = m
* depth (complete tree) = $\log_M N$
* each internal node in a complete tree has (m-1 keys)

Binary search tree | B Tree
* specialized M-ary
* each node has many keys

suppose tree btw two keys → x and y
values v → $x \leq v < y$.

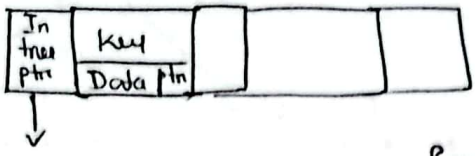
* each node takes full (page, blk, line) of mem (disk)

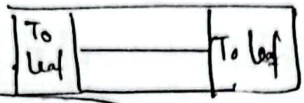
* root has 2-m children or at most m-1 keys.
* all other nodes has $(m/2) - (m)$ records
* keys + Data.

Objective of B/B+ tree:

Increase branching factor (degree) to reduce device access.

Btree:

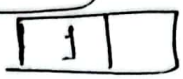




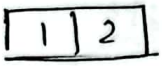
~~A A~~

degree - 3 = m → m-1 = 2 keys.

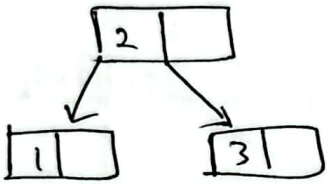
Insert 1:



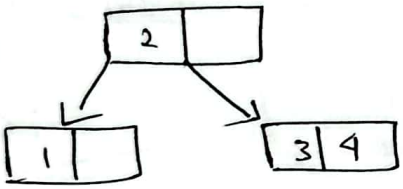
Insert 2:



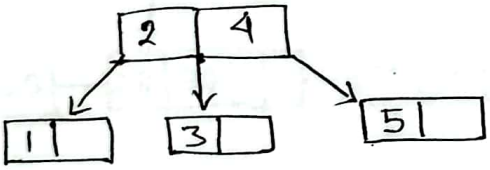
Insert 3:



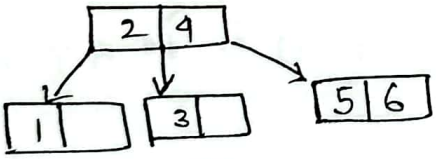
Insert 4:



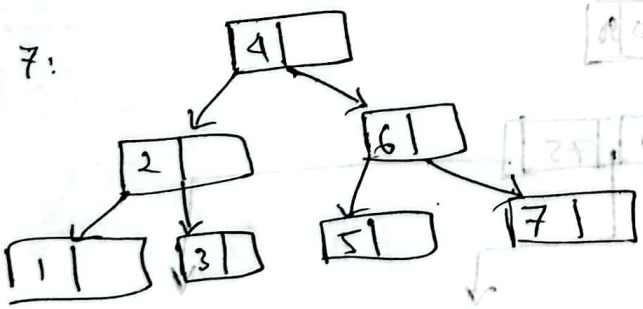
Insert 5:



Insert 6:



Insert 7:



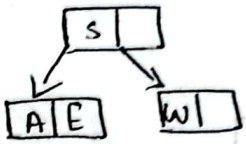
Practice: S W A E C T M L P Y X Q B

S:

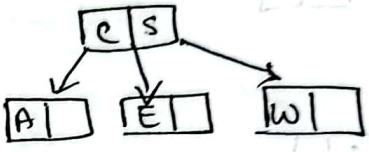
W:

A:

E:

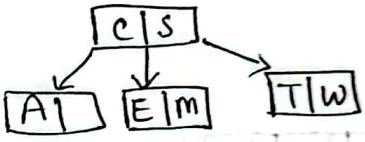


C:

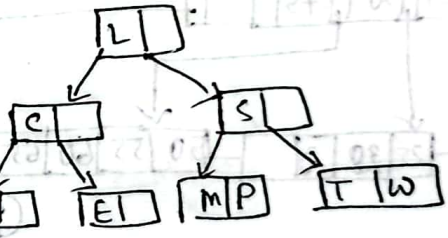


T:

M:

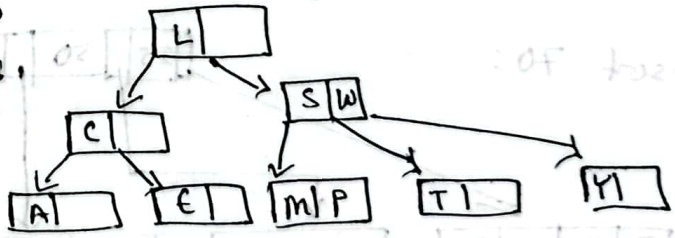


L:

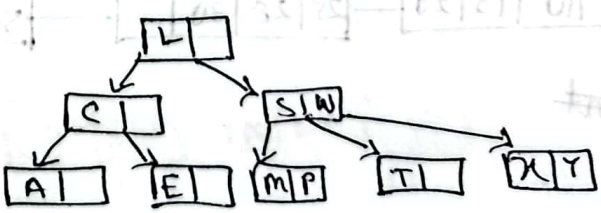


P:

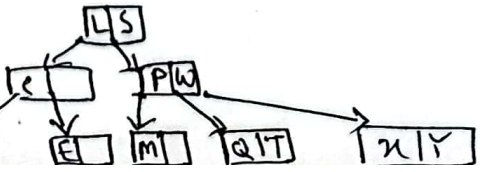
Y:



X:



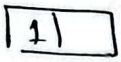
Q:



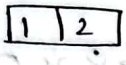
B:

B+ tree: Expl-1

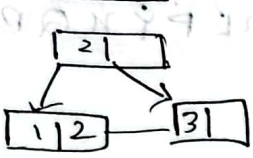
1:



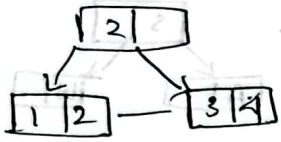
2:



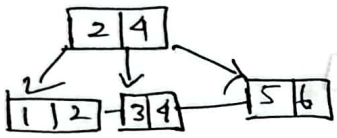
3:



4:

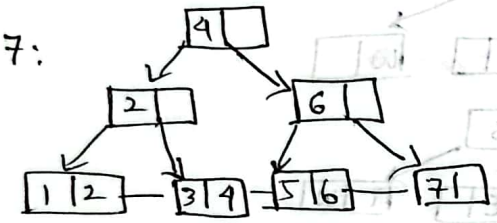


5:



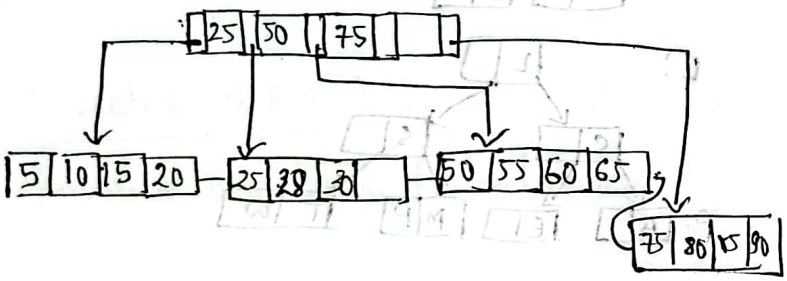
6:

7:

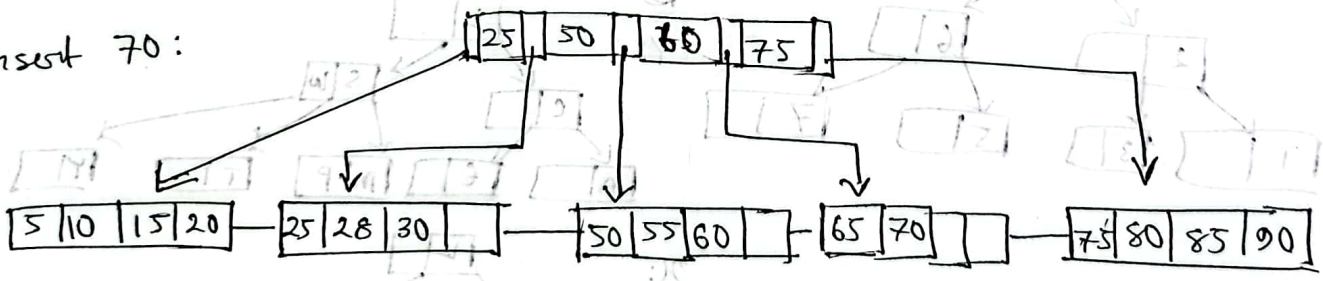


Expl-2

insert 28:

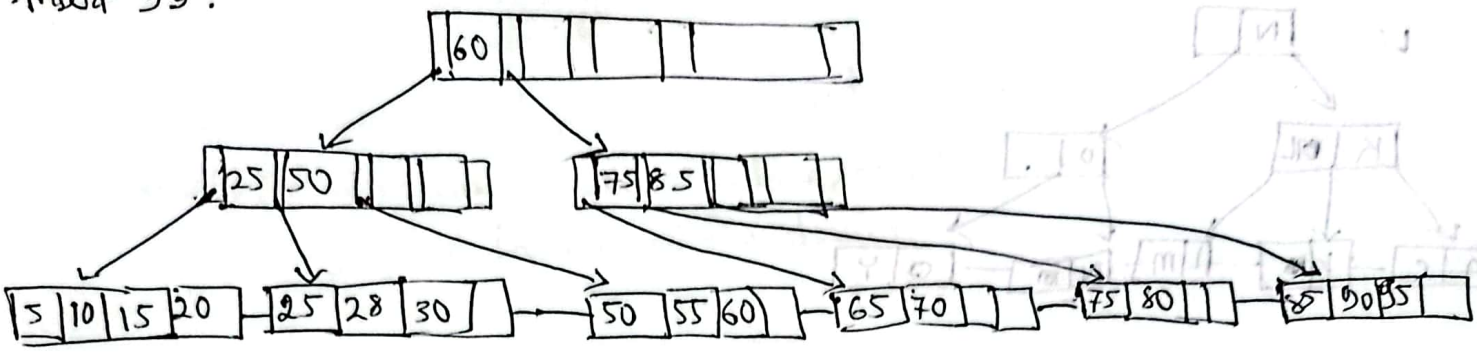


insert 70:

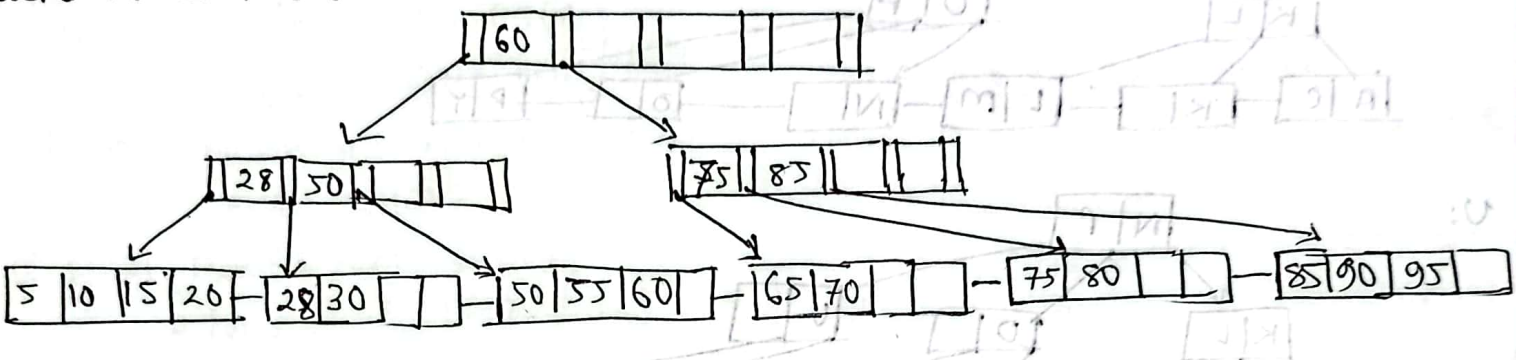


~~insert~~

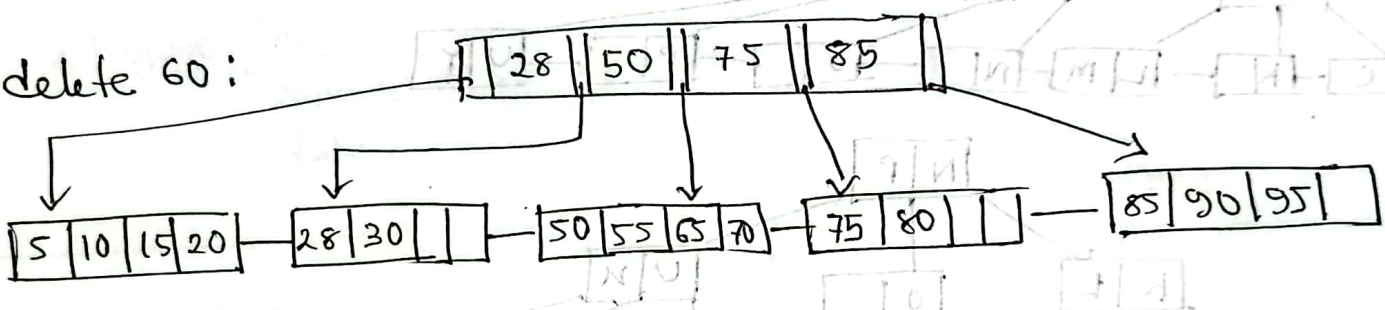
insert 95:



delete 70 then, 25:



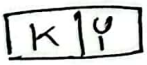
delete 60:



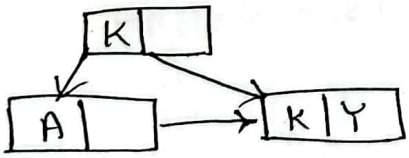
Practic: K Y A N C O M L P U X D F

K:

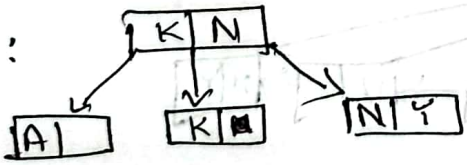
Y:



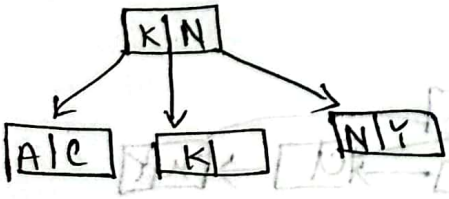
A:



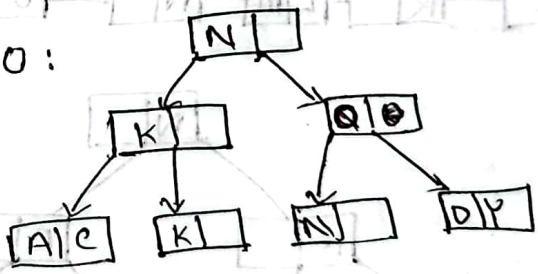
N:



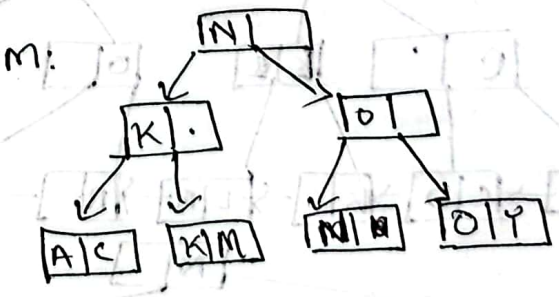
C:



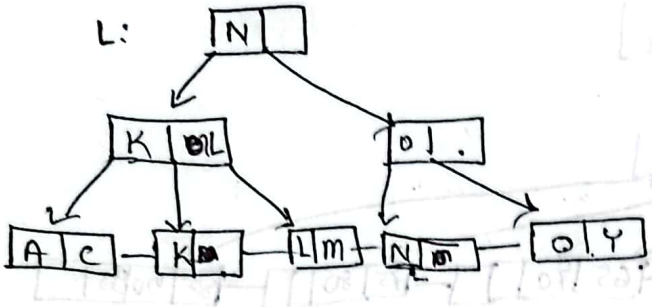
O:



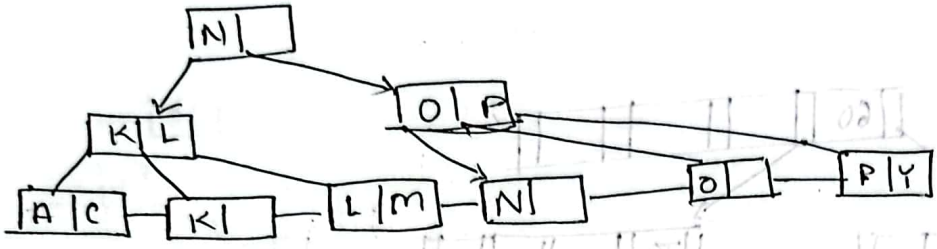
M:



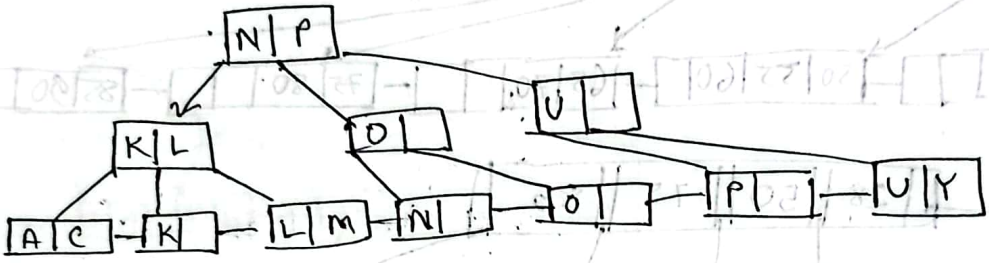
L:


N
K L
O
A C
K
L M
N
O Y

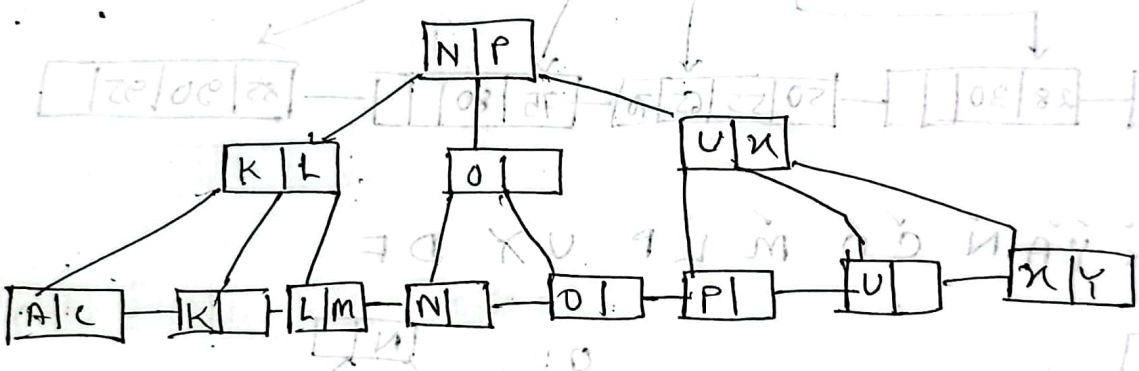
P:


N
K L
O P
A C
K
L M
N
O
P Y

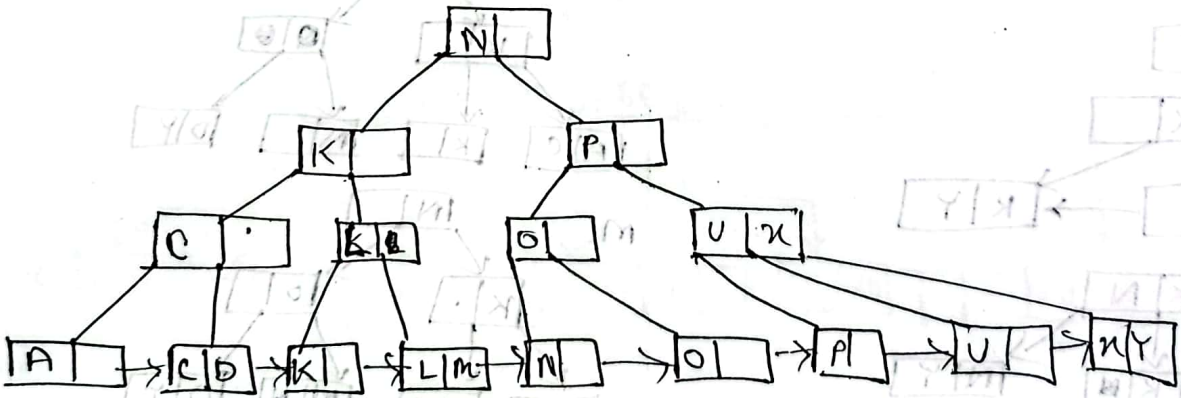
U:


N P
K L
O
U
A C
K
L M
N
O
P
U Y

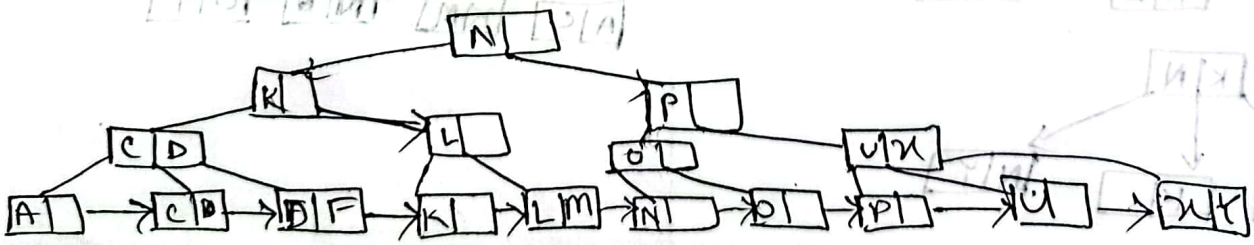
X:


N P
K L
O
U X
A C
K
L M
N
O
P
U
X Y

D:


N
K
P
C
K L
O
U X
A
C D
K
L M
N
O
P
U
X Y

F:


N
K
P
C D
L
O
U X
A
C D
D F
K
L M
N
O
P
U
X Y

STORAGE

Comparison: Speed, cost per unit of data, size.

Reliability: — data loss on power failure/crash
— physical failure of the device.

Memory

① Volatile (power off করলে data delete হয়ে যায়)

① Cache [processor + main memory র মাঝে থাকে]

② RAM [Random Access Memory
- temporarily holds data]

③ Register [CPU-তে থাকে
capable of storing small data / info]

② Non Volatile (power off করলেও থাকে)

① floppy disk (diskette).
- thin - super flexible (mag. disk)
- can store upto 360 KB.
- using double sided media increases to 720 KB capacity.

② compact disc (optical disc) 1982.
(CD)
- stores data in the form of pits and lands
- general purpose - 2 used.
- CD holds upto 700 MB.
- DVD. " " 4.7 GB.

③ HDD (Hard disk drive)
- stores digitally encoded data on platters (rapidly rotating disk)

④ SSD (Solid State Drive)
- store electronically
- not traditional.

⑤ Pendrive (small)/USB flash drive
- ultra portable.
- store/transfer data.
- uses semiconductor - solid state হয়.

⑥ SD Card. (Secure Digital Card).
- removable flash mem card.
- high capacity mem in small size

---

cache
↕
main mem
↕
flash mem
↕
magnetic disk (floppy)
↕
optical disk (CD)
↕
magnetic tapes

Storage Hierarchy types

- **Primary** (fastest, volatile)
  - cache mem
  - main "
- **Secondary / On Line storage** (Non volatile, moderately fast)
  - flash mem.
  - magnetic disc
- **Tertiary / off line** (lowest, non volatile, slow)
  - magnetic tape
  - optical storage

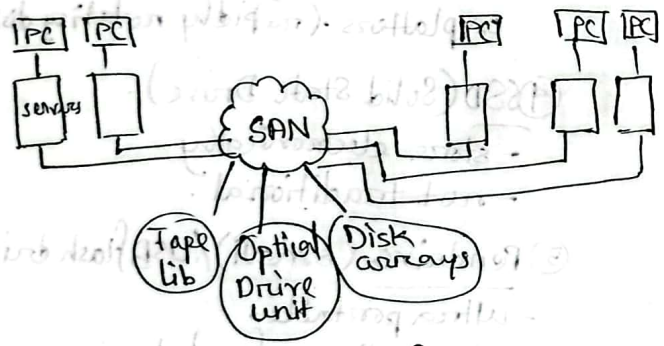
Disk Subsystem → directly comp.-এর সাথে connected থাকে

① Storage Area Networks (SAN) → অনেক গুলো disc connected
→ high speed network.
→ অনেকগুলো server.

② Network Attached Storage (NAS) → file system & interface.
→ network file sys protocol

SAN

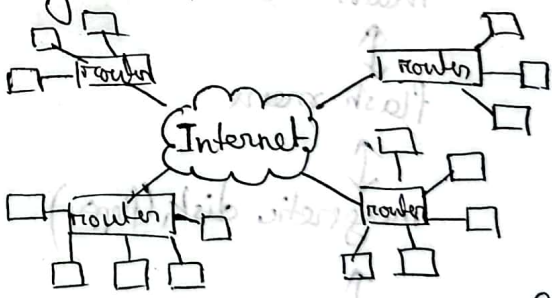
- allows multiple servers to access shared storage
- no need of traditional file server



⇒ Advantages: ① High Performance
② Scalability (expand করতে হয়)
③ Centralized Management
- efficient to allocate, money etc.

NAS

- file based – can be shared as files among multiple users/devices



⇒ Advantages: ① Ease of Use – user friendly.
- doesn't need advanced IT
② Cost Effective
③ Remote Access
④ Data Protection.
(includes RAID)

Components :-

SAN

① Hosts/Initiators :- Server/ devices accessing the storage

② Storage Devices/Targets :- disk arrays / tape Libraries

③ Fibre Channel : high speed ~~net~~ technology ~~just us~~ used in SANs

④ Switches : connects multiple devices in SAN

⑤ SAN software : includes management software, protocols for efficient data transfer/access.

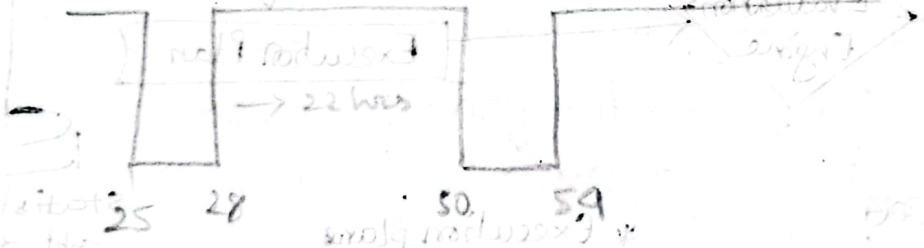
NAS

① NAS Device : physical/virtual that contains hard drives connect to net

② Net interface : ethernet port দিয়ে local net-এ connect করি.

③ OS : একটা embedded OS-এ run করি.

④ File system : data organization + access-এর জন্য লাগে

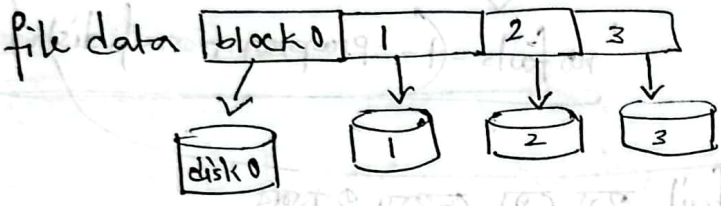


RAID

redundant array of Inexpensive ~~Data~~ Disks

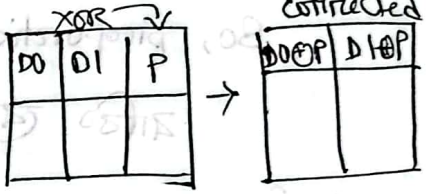
Purpose: Connects multiple disks so → Large storage
→ faster access
→ redundant data

Stripping: - take file data
- map it to diff disks
- allows for reading data in parallel.



Parity: - error checking & correction.

~~1~~ - ২য় প্রকার → even → 0
odd → 1
] XOR করব



Mirroring: - keep copies of data in 2 seperate disks.
- error recovery — if data lost gets from the copies
- Expensive — যেহেতু extra disk লাগে
- slow write performance — 2 বার সময় লিখা লাগে তাই
- read better — can read file in parallel.

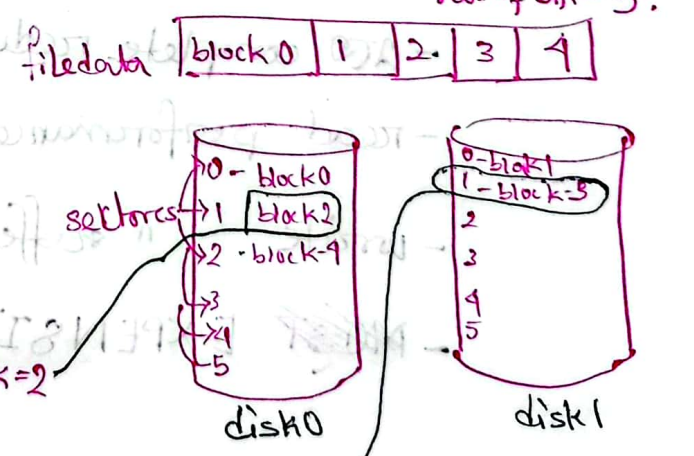
RAID - 0

- stripping
- file ⇒ block-এ ভাঙ্গে
- stripe blocks accross disk.
- simple implementation.

যেই disk-এ যাবে = file block % no. of disks
sector no. = file block / no. of disks

exple → disk no. = 2 % 2 = 0 for blk=2
sector no. = 2/2 = 1

or, disk no = 3 % 2 = 1 ; sector no = 3/2 = 1



- provides no redundancy / error detection.
[মনে] consider করা খুব imp cause — অনেক disks means low Mean Time To Failure (MTTF)

Failure Rate:

MTBF roughly proportional to no. of disks.

Proportion of disk fail যদি 5% হয় then

" " at least 1 fails = 1 - Proportion of no failures.

$= 1 - [1 - 0.05]^2 = 0.0975 = 9.75\%$

no fails = (1 - proportion of disk fails)

So, proportion of disk fail হয় তা দেওয়া থাকলে বাকি বের করব

Performance:

GOOD

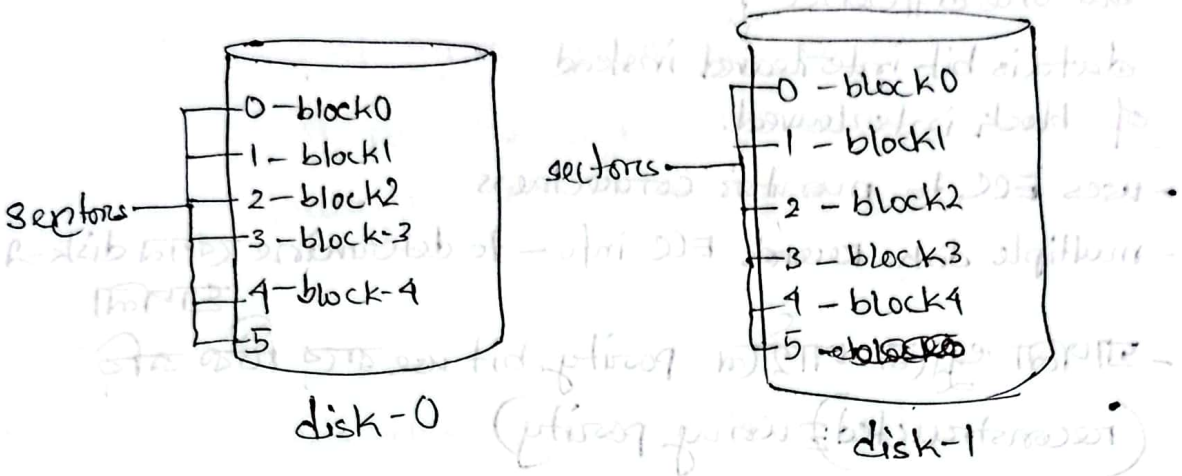
- fragments তাদের নিজেদের disk wise simultaneously same sector-এ লিখা থাকে যার জন্য parallel ভাবে পুরা data থেকে smaller section গুলা read করা easy হয়

RAID-1

- পুরা এক file একটা disk (single) রাখব.
- আরেকটা disk নিয়ে সেটাতেও copy করে রাখব.
- এতে complete redundancy পাওয়া সম্ভব
- read performance improve করা যায় [parallel-ভাবে read করি তাই]
- write " suffers - দুইবার write করে তাই
- EXPENSIVE RAID

দেখতে এমন → file data

block0	1	2	3	4



Failure Rate:

একটা fail করলে আরেকটা সমস্যা

কিন্তু দুইটা fail করার probability কত? 

Probability of a block failing in a disk = 5%.

" " the " " " both " = $(0.05)^2 = 0.25\%$

So, pr. of the block failing in both disk = (Probability of that block's fail is one disk)

performance:

improve করা সম্ভব

↓

each disk-এ 1 each independent disk controller use করব

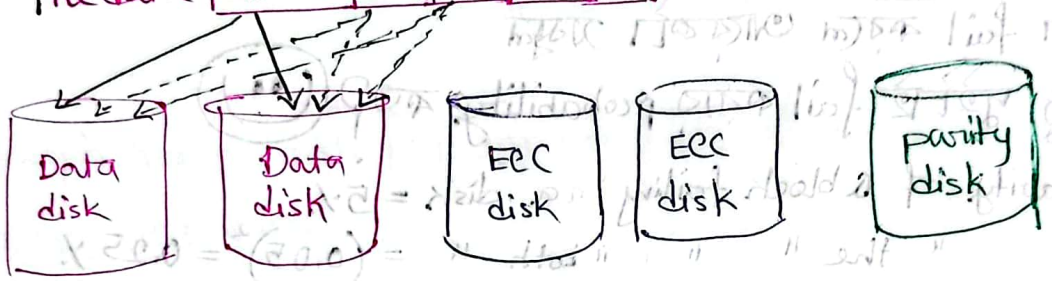
তাইলে read write parallel ভাবে করা যাবে - speed বাড়বে

(but expensive).

RAID-2

- stripes data like RAID 0.
  but one difference → data is bit interleaved instead of block interleaved.
- uses ECC to monitor correctness
- multiple disks records ECC info – to determine কোন disk-এ ভাঙলো
- ভাঙলো খুঁজে পাইলে parity bit use করে ঠিক করি (reconstructed using parity)

Visualize: file data | block 0 | 1 | 2 | 3 | 4 |



Reconstructing Data: suppose ৪টা disk.
correct data = 10011010
but read = 10011110 ; parity = 0 ;
disk-2 এ suppose সমস্যা (ECC করে ধরা হলো)
parity read data গুলোর সাথে XOR করে বের করব কোন bit flip করা লাগবে

Advantages: – fewer disks needed (not like RAID-1)

problems: ① performance – data + ECC-এর জন্য disk থেকে code পড়া লাগে
– write-এর আগে data modify, ECC, parity করা লাগে

② only one read at a time
– single block is read parallelly
– but bit interleaved-এর জন্য ~~≠~~ multiples blocks ~~files~~ from multiple files cannot be read at the same time.

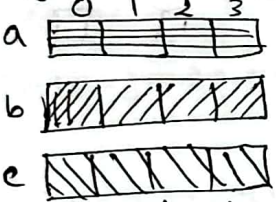
RAID-3

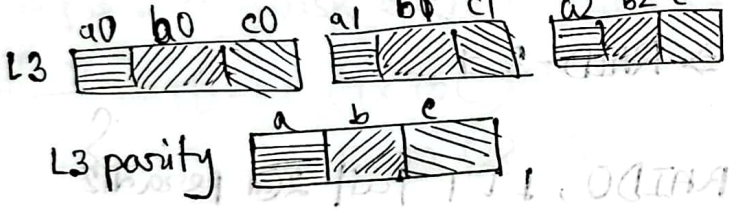
RAID-2 -তে অন্য disk use করে detect করা লাগে কোন disk-এ সমস্যা
RAID-3 -তে ECC code দেওয়া থাকে disk নিজেই বুঝে সমস্যা আছে কি না।
just parity disk লাগে correction-এর জন্য.
faulty disk নিজেই বলে দেয় parity disk কম লাগবে যে

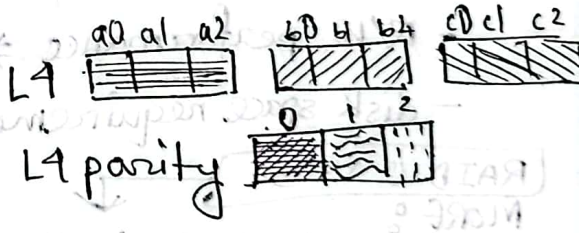
RAID-4

RAID-2,3 must access all disks to access a single file.
multiple I/O-এ parallelism নাই।
- but RAID-4 interleaves file blocks allowing that
- calculates parity over data from multiple blocks

If units →







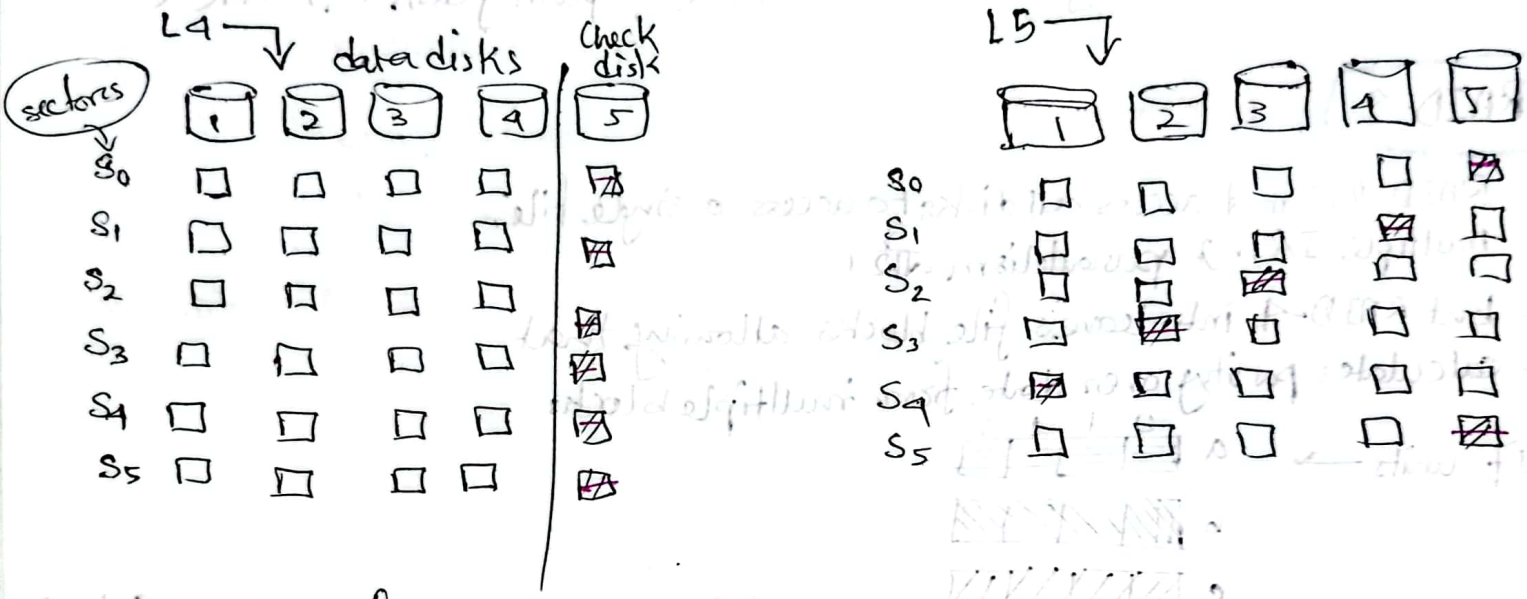
- RAID-4 simple - A block read করতে চাইলে disk0-তে থাকে problem পাইলে B,C,D, parity block দেখে ঠিক করে

- parity → new parity = (old data ⊕ new data) ⊕ old parity.
- a write requires - 2 reads 2 writes

performance
- multiple small reads are faster
- writes খুব slow (parity calculation + write এক সাথে তাই)
- 1 write at a time (check disk-এ আগে কেও নাই wait করা লাগে).

RAID-5

- stripes + checks all the disks
- no longer single check disk
- improves multiple writes (parallelism)
- " reads (one more disk use for read)..



- R/W performance ⇒ RAID1
- disk space requirement like ⇒ RAID-3,4.

RAID 0 vs 5

MORE :

① MTBF better than RAID0, 1 disk fail हो तो पता नहीं but 2 disk fails need time (more)

② Good performance Like RAID 0. not better.
R/W — parallelism.

③ Drawbacks ! → heavy parity calculations. अतः W(operation) is slower compared to RAID 0.

④ Useful where reads are frequent than write.

RAID 10 → Combines RAID 0 + RAID 1

- Highest performance
- most expensive
- Striping + Mirroring exists
  - ↓ gives R/W performance
  - ↓ gives best redundancy

Storage Capacity formulae : (SC) [i → level] (যেটা প্রথমেই লিখেছি note-এ, মনে)

(*) RAID-0 : no redundancy → sum of capacities of all disks

$SC_0 = (\text{no. of disks}) \times (\text{capacity of each disk})$

(*) RAID-1 : mirroring exists · so ⁂ twice the capacity of 1 disk.

$SC_1 = (\text{capacity of each disk}) \times 2$.

(*) RAID-2 : like RAID-0 .

(*) RAID-3 : 1টা disk must parity হবে.

$SC_3 = (\text{no. of disks} - 1) \times (\text{capacity of each})$.

(*) RAID-4, 5 → same as RAID-3

(*) RAID-10 → RAID-1 - এর মতো .

Example : 4টা disk each 750GB. Calculate SC for RAID-0, 1, 5.

Soln :

RAID-0 : (4×750) GB

$= 3000$ GB $= 3$ TB

RAID-1 : (2×750) GB.

$= 1500$ GB $= 1.5$ TB.

RAID-5 : $(4-1) \times 750 = (3 \times 750)$ GB

$= 2250$ GB $= 2.25$ TB