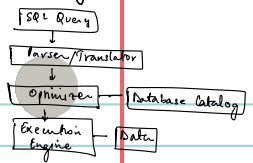
Query Execution in SQL Databases



Parser → Checks – Syntax, Verify Query
SQL Query → Relational algebra i.e. High-level → Low level

Optimizer → Chooses most efficient execution plan considering factors such as Indexes, Join & Data distribution

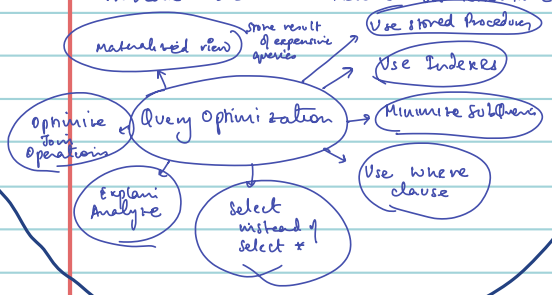
Execution → Query is executed and the data is retrieved based on the order of execution of given Query.

Order of execution

1. From
2. Where
3. Group by
4. Select (~) from
5. Order by

Query optimization in Relational Databases.

Goal: Reduce the load on system Resources & Provide accurate Results in lesser time



MV:
physically the data copy of select result
not fresh data

NoSQL → Use non primary, composite (multi field query) inverted index (for full & elastic text search)

Indexing

Index - Data structures that improve speed of data retrival in DB.

Fast for Queries having: select, Join, where, order by

Typically implemented as trees, bitmaps, hashmaps for quick lookups

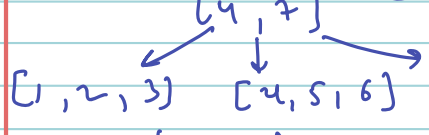
Occupy additional disk space

Index types

B+ Tree

- Balanced tree DS
- Fast range, between, & ordered query exec.
- Internal nodes: Contains Keys & pointers to child node
- Leaf nodes: contain pointers to actual data
- All leaf are at same level. making balanced tree.

Table employees: EmployeeId's [order=3]



start → node
move to child node: [4, 5, 6]
locate 5 leaf node

→ Used in MySQL, Postgres

Bitmap

- Ideal for columns with low cardinality ie less distinct values
- Uses bit array to represent column values (married, gender)

id	gender
1	m
2	f
3	m
4	m

bitmap m: 1011
f: 0100

bits = no. of rows

for fast read and efficient combination of multiple conditions

hash map

DS used in map

Used in in-memory databases, example Redis

Compose of: Key, hash function & hash table

fast for equality = operations O(1)

① Use hash function to compute index from key

② Store the value associated with key at the index determined by hash()

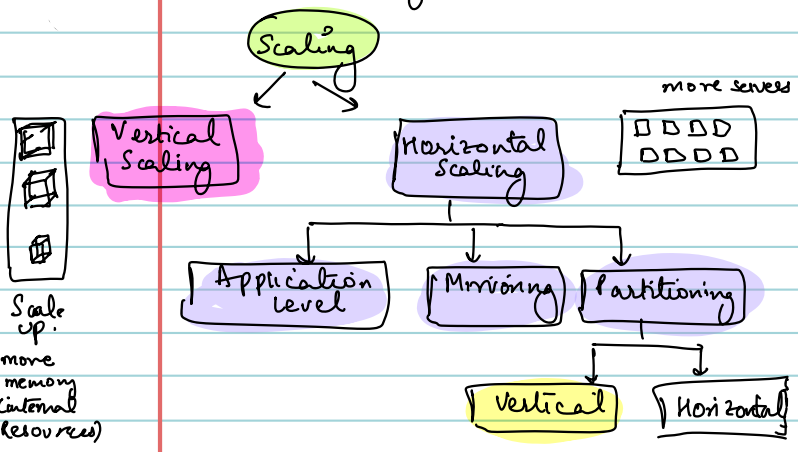
- not good for range ()

MapReduce

key with values using hash().

Database scaling and big data processing. Describe different approaches to database scaling (Vertical and Horizontal scaling, application level, mirroring, partitioning, sharding) and non-database approaches to big data storing and processing (DW, DataLakes, ETL/ELT).

Database Scaling



Vertical scaling: Single server → ↑ Resources
- RAM
- CPU
- Storage (HDD)

to handle ↑ load.

- Typically used with Relational DB MySQL, PostgreSQL.

Horizontal scaling: Multiple server → load distributed across several servers & each server works on part of the workload
(↑s capacity by adding ↑ nodes)

- Most common in NoSQL databases (Cassandra)
- SQL DB → possible with sharding
- In SQL, less common due to consistency requirements.

Application level scaling: Implemented at software (application level) → to help manage ↑ load
- Performance
- demand

w/o scaling directly

Idea: to reduce Burden on DB, and do handling of some tasks at the application layer

- In both SQL and NoSQL

eg: Redis is used for cache (NR, NSQL)

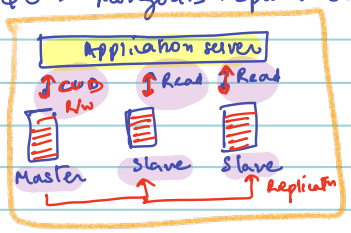
Mirroring: Keeping copies of database on multiple servers to ensure high availability.

SQL → MySQL → for Fault Tolerance

NoSQL → MongoDB Replica sets.

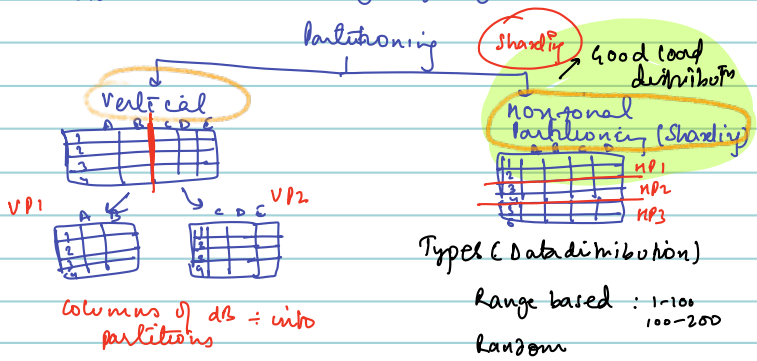
Appln: Post response for R Queries

Slow slave knows the invalid db state



Partitioning: Divide large database into smaller manageable pieces · each stored on different server

Physically dividing data into separate data stores → To improve performance scalability of large databases



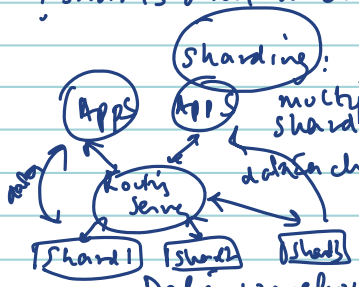
Columns of DB ÷ into partitions

Types (Data distribution)
- Range based : 1-100, 100-200
- Random
- Data innerlogic
- Geospatial location
- Math function
- Relationship b/w shards (servers) don't exist or Management to route to shards

Partition: ÷ within same database

Sharding: ÷ among multiple databases or across servers

! shards don't interact



Data warehouse: Central repository to store large amt of structured data from multiple sources. Optimized for query and analysis & transaction process

employs: Star or Snow flake schema

User OLAP for complex queries

Data lake: large repo. to store unstructured, semi structured, structured data. Stores data in native form

Schema on Read: Data interpreted

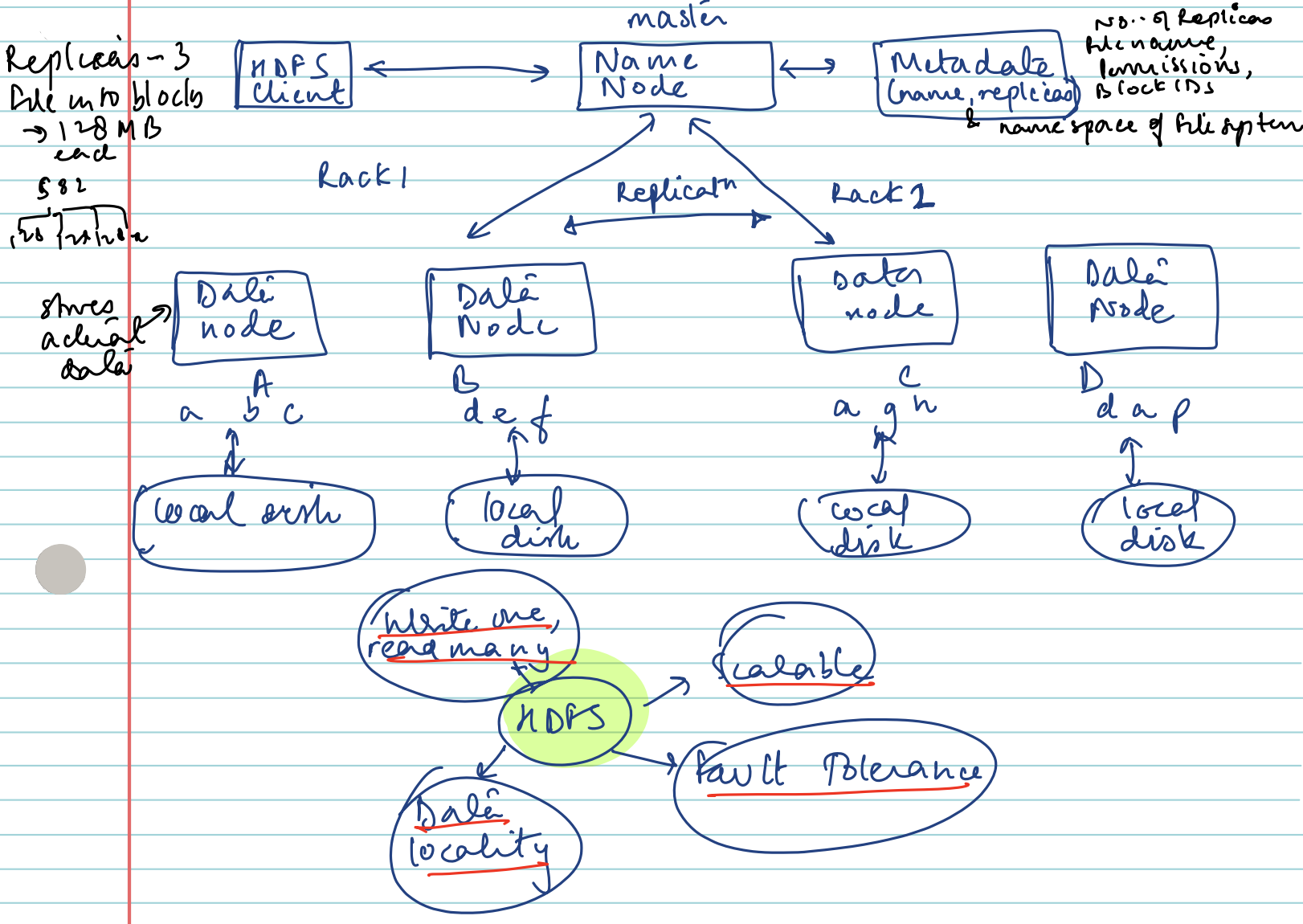
eg: AWS S3, Azure Data Lake

ETL	ELT
• for structured	all data types
• small data set	• large data sets
• slower	• faster (since it loads data without transforming it)

At storage layer → Distributed file storage

HDFS Architecture: Hadoop distributed file system

In Big data (huge data), it is designed to handle large volume of data across distributed N/w of computers.

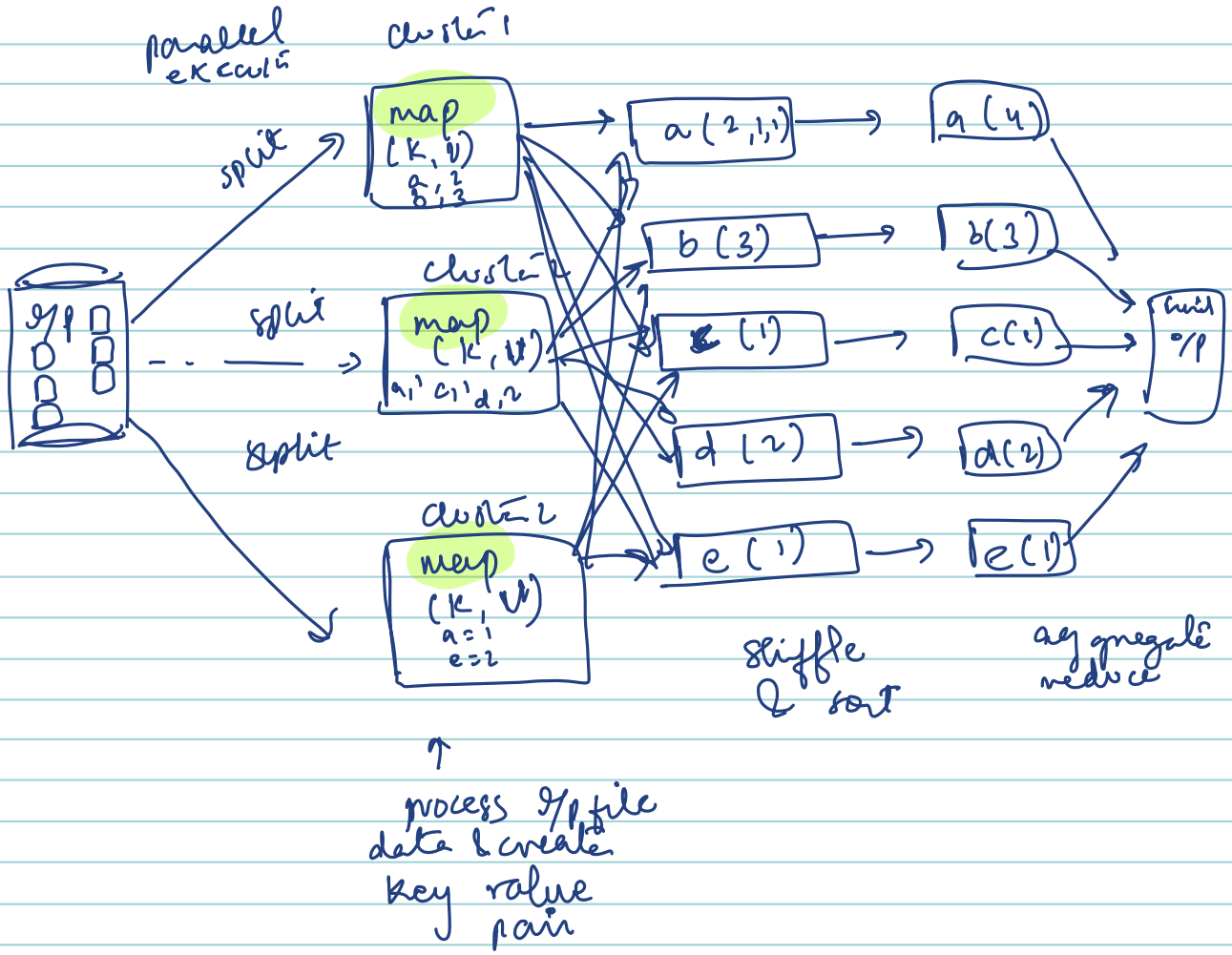


Application layer = Hadoop Map Reduce → for processing data

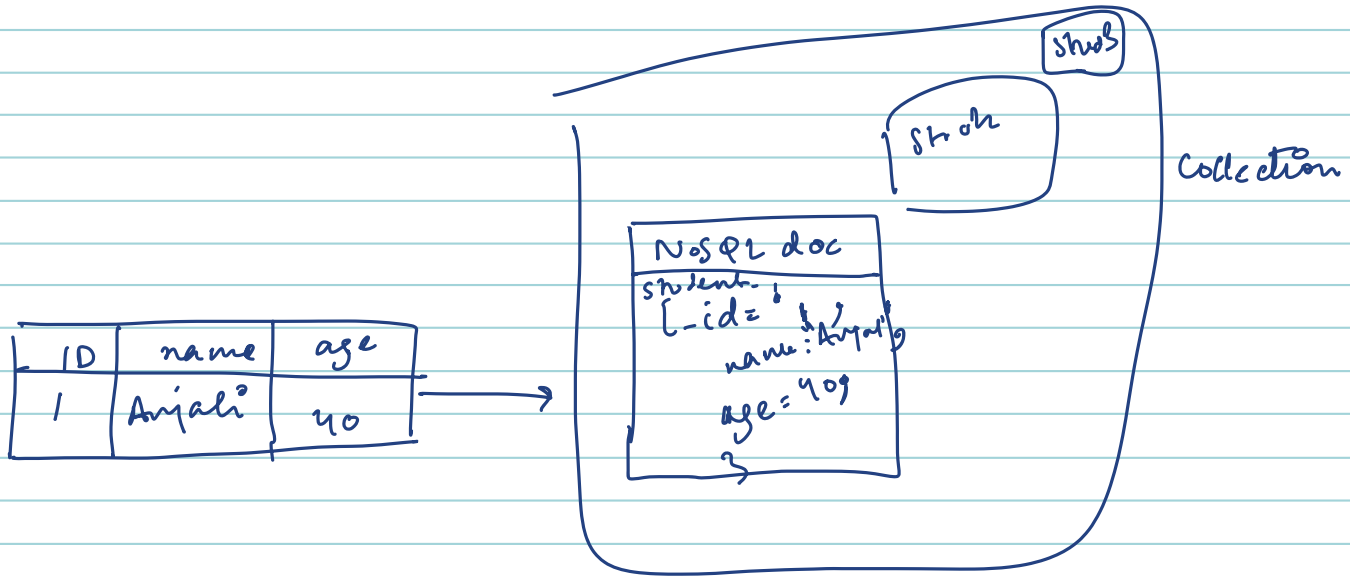
Map Reduce → Data processing technique for large datasets in D & parallel manner.

- Used in Hadoop widely.
- Simplifies data processing across cluster of machines by ÷ tasks into sub tasks and executing them in parallel.

Count the occurance of words in a document

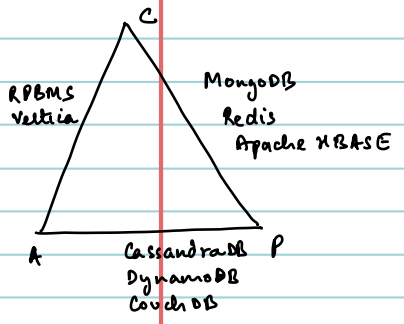


c



AP Theoram [Brewer's Theoram]

C = Consistency → All nodes see same data all the time
A = Availability → Each request receives a response (recent or not)
P = Partition Tolerance → System continues to operate despite n/w partitions of comm^n failures b/w nodes



"For a distributed System, It is impossible to achieve all 3 (CAP) simultaneously.

During Partition Tolerance (Network failure), either Consistency or Availability can be provide by a system (DB).

ACID

- Atomicity
- Consistency
- Isolation
- Durability

} properly to ensure reliable transaction processing

Txn is

Atomicity: "All or Nothing"

If any part of txn fails the entire transaction is rolled back

Txn
Account A → Account B & no changes are applied.
100 80 80 (CN of updated) → Rolled back
→ sent 100

Consistency: follows rules

brings DB from one valid state to another valid state

eg: -ve balance (Rule Violation)
Sum of total remains same

Isolation: Concurrent Txns are isolated from each other.

T1 (RCA) | T2 (RCB) | T3 RCA).

Customer withdrawing money from same account, isolation ensures that both the txn don't see each other.

Schedule
[Serial schedule] → It is always consistent

Durability: All changes in DB should be permanent even if system fail

replication

[SQL]

BASE Properties

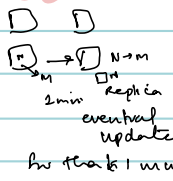
NoSQL

- BA → Basically Available
- S → Soft State
- E → Eventually consistent

Basically Available: System Guarantees availability.
DB will respond to all requests even if data sent is stale or incomplete

eg: Inventory System – sold out not updated on website during high traffic

Soft state: Even w/o Input from outside, the state of system changes. Due to the eventual updates across distributed system



Eventually consistent: System will eventually become consistent given enough time.

For update it will make time (replicas) But eventually become consistent.

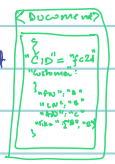
Types of NoSQL Databases

Key value

- Based on key value principle
- Data accessed only using key (Index)
- least Complex
- Schemaless (Flexible) (can remain empty)
- Example: Redis, Dynamo DB Amazon

Document Database:

- Extension of key value
- Value is stored in document format (XML or JSON)
- A complex than k-v pair
- key could be number, character
- Document with same structure are grouped into collections.



example → MongoDB, Couch DB

"Best for storing catalogs"

Column Database: work on column

ID	Name	No
1	A	22
2	B	33
3	C	44

On row DB (SQL) = 1, A, 22; 2, B, 33; 3, C, 44

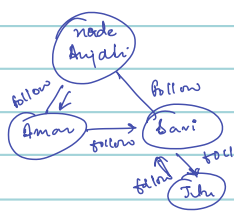
Data warehouse → In column DB = 1, 2, 3; A, B, C; 22, 33, 44

Faster lookups

Avg min, max analytics special

Cassandra: example

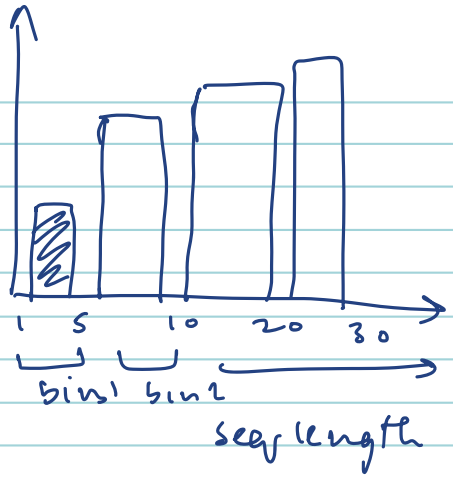
Graph Database: For highly connected data eg: Social Media GraphDB is used



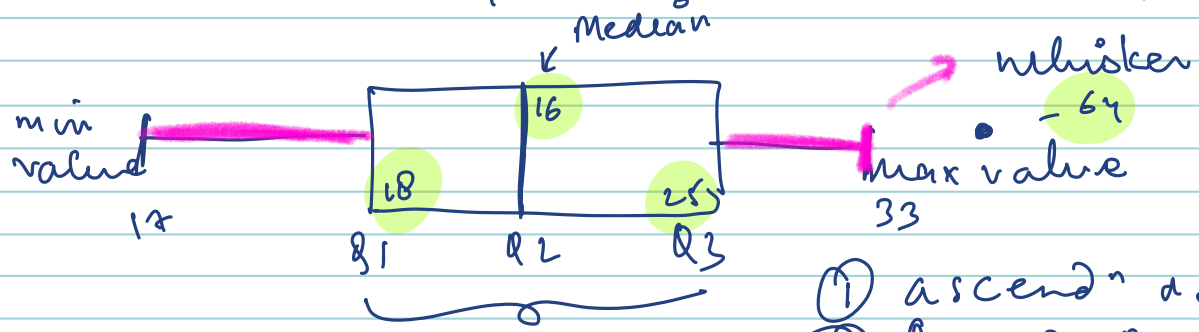
Example: Infinite Graph

histogram:

- distribution of data by freq ÷ them into bins
- bin → show frequency of data points



Box plot: show dist of data (how data is spread)



① ascend^n data
② Q_2, Q_1, Q_3
③ Lower & higher outlier

---

data in asc format

17
18 — Q1
20
22 — +1/2
23
25 → Q3
33
64

$\frac{n+n}{2} = 42/2$

→ 16 → Q_1 Median

Q_1, Q_2, Q_3
↑
Median

→ Check Outlier **higher outlier**

$= Q_3 + 1.5 * IQR$

$=$

$IQR = Q_3 - Q_1$

$(25 - 18)$

$= Q_3 + 7 \times 1.5$

$25 + 10.5$

$= 35.5$

any data > 35.5 would be outlier

∴ 64 is outlier

② Lower Outlier

$= Q_1 - 1.5 * IQR$

$= 18 - 10.5 \Rightarrow 8.5$

No lower outlier

Covariance

$$Cov(X,Y) = \sum_{i=1}^{n} \frac{1}{n-1} (X_i - \mu_{xi})(Y_i - \mu_{yi})$$

(sample)

Indicates direction of the relation btw variable

$-\infty$ to ∞

	ht	wt (kg)	ht	wt (g)
cov:	143		143153	→ 1000 x longer (not stronger though)

→ can't tell about strength only show +ve or -ve

calculates strength or strength

correlation

$$Corr(X,Y) = \frac{Cov(X,Y)}{\sigma(X)\,\sigma(Y)}$$ (SD)

- Indicates strength & direction of relation btw variables

ht	wt (kg)	Ht	wt (g)
	0.922		0.922 → same correlation

$-1 < Corr(X,Y) < 1$

doesn't influence by scale: Standardises data

if 0 = no relat
- 1 → strong negative
1 → strong positive