DBMS :-

1) Data
2) Types of Data → Qualitative, Quantitative
3) Information
4) Database
5) DBMS
6) DBMS Vs File System
   - Data Redundancy
   - Data Isolation
   - Data Inconsistency
   - Data Integrity
   - Atomicity
7) DBMS Architecture
   - View of Data (Three Schema Architecture)
     - Physical/Internal level
     - Internal mapping
     - Logical/Conceptual level
     - External mapping
     - External/View level, External view level
8) Instance
9) Data model
   - ER, Object-oriented, relational, object-relational
10) Database Languages
    - DDL, DML → SQL
11) How is Database Accessed from Application Program?
12) Database Administrator (DBA)
13) DBMS Architecture
    - Three Schema Architecture
      - View of data
    - Application Architecture
      - T1, T2, T3
      - API → ODBC, JDBC
14) Entity
15) Types of Entity
    - Strong (with P.k)
    - Weak (No P.k)
      - LOAN → gives → Payment
16) Entity set
17) Attributes (domain, value)
18) Types of Attributes
    - SCMDNS
      - Simple
      - Composite
      - Multi-valued
      - Derived
      - Null
        - Not known
        - Unknown
      - Single-valued
19) Relationships
    - Strong
    - weak
    - Degree of Relationship
      - unary
      - Binary
      - Ternary

Relationship Constraints
- Mapping Cardinality
  - One-to-one
  - One-to-many
  - many-to-one
  - many-to-many
- Participation Constraint
  - Partial Participation
  - Total Participation

eg: Customer Borrows Loan

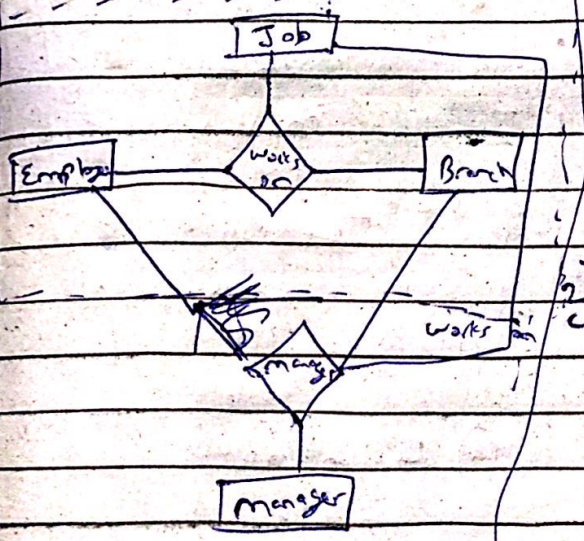
20) ER features
- Specialisation (is-a)
- Generalisation

Attribute Inheritance

*) Participation Inheritance in these features

21) Aggregation
- Relationship among relationship

eg



22) Relationship model
- Describe the Association through tables

23) Degree of table
- No. of Attributes or columns

24) Cardinality of Table
- No. of Tuples

25) Relational Key
- A unique set of Attribute which can uniquely identify a tuple

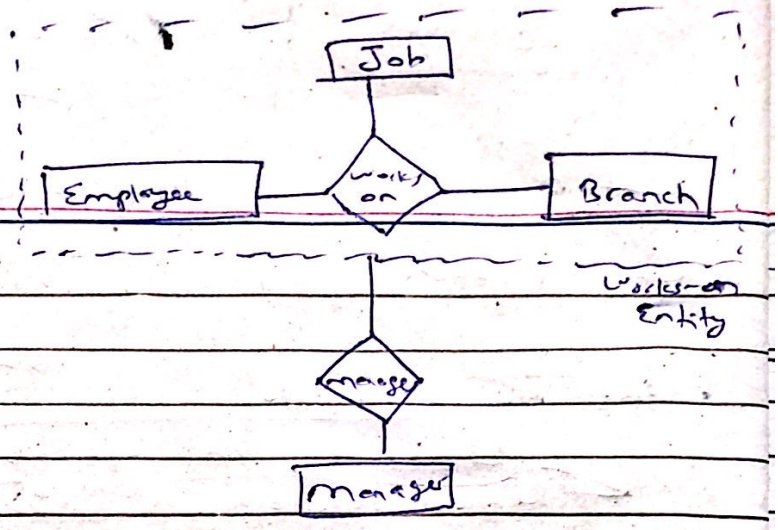
26) Keys
- SCPAFCCS
  - Super Key
  - Candidate Key
  - Primary Key
  - Alternate Key
  - Foreign Key
  - Composite Key
  - Compound Key
  - Surrogate Key

27) Integrity Constraint
- DEKR
  - Domain Constraint
  - Entity Constraint
  - Key Constraint
  - Referential constraint

Tuple of FK must be present as PK in referenced table or it can be null

Referential constraint

- Insert constraint
  - Value can't be added in referential (child) table unless it is present in referenced (parent) table
- Delete constraint
  - Value can't be deleted from parent table if it is being used in child table
  - (ON DELETE CASCADE)
  - Value of F.K must be present as P.K in parent table or it can be null
- ON DELETE NULL
  - When entry from parent table is delete then ~~delete~~ null the value corresponding in child table.



2B) Normalisation :-

- Functional Dependency (FD)
  - A → B
  - A: Determinent, B: Dependent
- Types
  - Trival
    - A ⊇ B ⇒ A → B
    - A → A , B → B
  - Non-Trival
    - A → B , B ⊄ A

Key Constraint :-

- NOT NULL
- UNIQUE
- DEFAULT
- CHECK
- P.K
- F.K

- Rules of FD
  - Reflexive
    - B ⊂ A then A → B
  - Augmentation
    - A → B then Ax → Bx
  - Transitivity
    - A → B , B → C then A → C

Remove Redundancy

- ~~Anomalies~~ Why Normalisation?

Redundancy create Problems

- Insertion
- Deletion
- Updation

What Normalisation do?

- Reduce till Single Responsibility Principle
- Remove redundancy by dividing larger table to smaller
  eg student, Branch, course

Types of Normalisation

- 1NF
  - Value must be atomic

eg

Employee

id	name	Phone
1	A	88
2	B	12,99

By atomic mean handle multi-values

id	name	Phone
1	A	88
2	B	12
2	B	99

- 2NF
  - Relation must be 1NF
  - No Partial Dependency

eg

Relation (A B C D)

{A B} → Composite P.K

In this case

A, B → Prime Attribute

C, D → Non-Prime Attributes

AB → C ✓

AB → D ✓

But

B → C ✗ Partial Dependency

Non-Prime attribute can be determine by part of prime attribute

2NF conversion

- R1 (A B D) AB → D
- R2 (B C) B → C

A	B
Null	1
2	Null
Null	Null

How B → C going happen

3NF

- Relation must be 2NF
- No transitivity
  - i.e Non-Prime attribute can't find non-prime attribute

eg

Relation (A B C)

In this case

A → Prime Attribute

B, C → Non-Prime

Here,

A → B

B → C

We can see transitivity here Here, B can determine C which we are going to omit

3NF conversion

R1 (A B)

R2 (B C)

Table :-

Before 3NF

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

Redundancy

After 3 NF

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

B	C
1	x
2	y
3	z

Boyce-Codd-Normal-Form (BCNF)

→ Relation must be 3NF

→ Any prime attribute must not be determined from prime or not prime attribute.

Example

Std-id	Subject	Professor
101	Java	PJ
101	CPP	PC
102	Java	PJ2
103	C#	PC#
104	Java	PJ

Here observation is

→ One Professor can teach only one subject

→ One subject can be taught by more than one professors

{Std-id, Subject} → PK

So;

{Std-id, Subject} → Professor

Professor → Subject

Here, Professor being non-Prime attribute is determining subject i-e prime.

BCNF Conversion :-

(1)

Std-id	Prof-id
101	1
102	2
⋮	⋮

(2)

Prof-id	Professor	Subject
1	PJ	Java
2	PC	CPP
⋮	⋮	⋮

1) Transaction

↓

Lecture 13

1) Indexing

Why indexing?

↳ Optimize performance of DBMS

Operations speed up

↳ SELECT, WHERE

Operation slowed down

↳ Write operation, update, delete

Based on non-key attribute

↳ Example GROUP BY

⇒ Based on key attribute (P.K)

↳ Sparse Indexing

⇒ Based on Non-key Attribute

↳ Dense Indexing

↳ Can have duplicate values in disk so use dense indexing

↳ GROUP BY

↳ Clustering indexing

→ Primary Indexing } Binary search.

↳ Sorted datafile

↳ Dense & Sparse indexing

↳ Based on key Attribute

↳ Based on Non-key Attribute

↳ multi-level indexing

↳ Clustered Indexing

⟶ Secondary Indexing

↳ Unsorted Datafile i-e search key is unsorted

How unsorted?

↳ Bcz if a datafile is sorted based on rollno then it will be automatically unsorted based on age

↳ Dense Indexing

↳ Benefit of indexing in Secondary is that the indexing is sorted that means we can use Binary Search on index table.

↳ In case of duplicates we can store linked list as data reference against search key in Index table

*) NoSQL Database

- Not Only SQL
- Flexible schema
- Structure, Unstructure, Semi-structure data handling
- Scaling
  - Horizontal
  - Vertical
- Non-Relational
- Focus is that if data is redundant then no problem but data access must be fast (Chats)

In SQL

- No Horizontal Scaling
- Horizontal Scaling
  - Achieved through Sharding or Replica-set
    - Multiple Servers contain some or copies of data

- High Availability
  - Due to Sharding

- Advantages
  - Flexible schema
    - In SQL you gotta store null values as the schema is predefined.
    - Here in flexible schema you can omit whatever is not available (JSON)
  - Horizontal Scaling
    - Add Additional node or CPU or system
  - Vertical Scaling
    - Update RAM, Storage, Hardware, CPU on current system
  - Easy Insert & Read operation
    - Due to all data present at one place on the other hand in SQL joins can be expensive
  - Difficult in delete & update due to all data load in memory and access and then modified in NoSQL
  - Caching mechanism

Type of NoSQL

- Key-Value Store
  - MongoDB
- Column Oriented
  - Benefit in Aggregate operation (max, min)

Name 1	Name 2	Age 1	Age 2

- Document Based
  - JSON
  - Support ACID
  - mongoDB
- Graph-Based Schema
  - Facebook

Types of Database

- Relational Database
- Object-Oriented Database
  - All bits of info in one package object instead of multiple tables
  - Doesn't support Views
  - High complexity in CRUD due to methods running time
- Hierarchical DB
  - File System
  - Disk Storage System
  - Easy to traverse
  - Website dropdown
- No SQL
- Network Database
  - Graph
  - M:N Links

No-SQL

- Doesn't support ACID In general, mongoDB can but not all of them do
- Don't have their own consistency constraints
- Data Redundancy
- Update, Delete is costly

SQL Vs No SQL

- Schema
- Scaling
- ACID
- JOINS (Rare in SQL and not in NoSQL)
- Purpose
- Example
- History
- Data Model (Fixed in SQL)

Clustering / Replica-Set

↳ Cluster → set of servers
↳ Replica sets

↳ Advantages
- ↳ High Availability
- ↳ Load Balancing
- ↳ Abstraction
- ↳ Redundancy

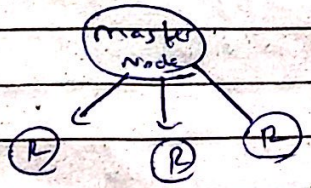
Partitioning & Sharding

↳ Data Huge & no. of req large
↳ Manageability

solution

~~Distributed~~
Distributed Database

↳ Database Optimization
- ↳ Scaling
- ↳ Add-Replica set (Clustering)



↳ Partitioning
- ↳ Scale-out (Horizontal Scaling)
- ↳ New nodes added

↳ Types of Partitioning
- ↳ Horizontal (Slicing rows)
- ↳ Vertical (Slicing cols)

↳ Advantages of Partitioning
- ↳ Parallelism
- ↳ Performance
- ↳ Availability
- ↳ Manageability
- ↳ Reduce cost
  - i.e Vertical scaling cost

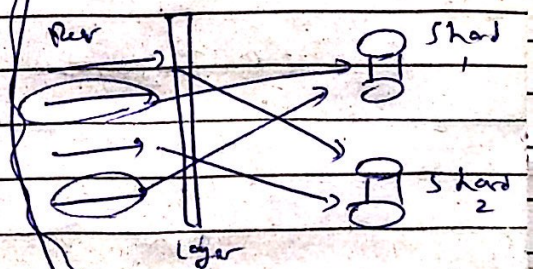
⊛ DB Optimization Techniques

↳ Clustering
↳ Partitioning
↳ Sharding

⊛ Sharding

↳ It is a technique to implement Horizontal partitioning

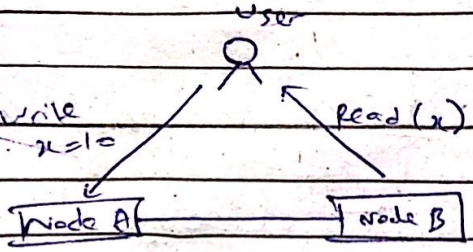
↳ Add Routing Layer



↳ Cons
- ↳ Non-Uniformity
  - ↓
  - Re-Sharding
- ↳ Problem in Analytic
  - i.e Aggregation Techniques

CAP Theorem

- Consistency, Availability, Partitioning Tolerance
- Only two among these three can exist at a time in dB
- Example
  - Only true for distributed db system
  - Three Kinds
    - CA Database
      - Practically impossible bcz in distributed system, partition occurs always
    - CP Database
      - When partitioning occurs then the NOPE B will get turned off until partioning is fixed. The ensure data consistency
      - Banking System
    - AP Database
      - When we have that system can be consistent after some time but it should be available. i.e we have that when partioning occurs then system can be available after some time when partioning gets fixed
      - Example
        - Facebook friend req



A Master-Slave Architecture

- Allow Replication happens??
  - Asynchronous
  - Synchronous
    - Async
      - Example → Fb chats
      - Delay can happen
    - Sync
      - Example → Bank
      - Data updation must be performed on all replicas or slaves before message "Data updated successfully"
- Master node
  - Write operation handling
- Advantages
  - Backup
  - Scale-out-read operation
  - Available
  - Latency reduce
  - Parallelism

Reduced single point of failure by
introducing replica db